NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

FRONT LOADED ACCURATE REQUIREMENTS ENGINEERING (FLARE); A REQUIREMENTS ANALYSIS CONCEPT FOR THE 21ST CENTURY

by

Anthony E. Leonard

June, 1997

Thesis Advisor:	Luqi
Co-Advisor:	Valdis Berzins

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. FRONT LOADED ACCURATE REQUIREMENTS ENGINEERING (FLARE); A REQUIREMENTS ANALYSIS CONCEPT FOR THE 21ST CENTURY	5. FUNDING NUMBERS
6. AUTHOR(S) Leonard Anthony E.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis focuses on ways to apply requirements engineering techniques and methods during the development and evolution of DoD software systems in an effort to reduce changes to system requirements. The major goal of this thesis is to provide a feasible course of action (COA) that reduces changes to requirements caused by the turnover of DoD decision-makers.

We demonstrate a distributed requirements engineering environment using computer aided software engineering tools linked together with electronic mail. We create this distributed requirements engineering environment using Netscape Communicator, Microsoft's Internet Explorer, Microsoft's Access97 database, Rational Corporation's Rational Rose, Matt Wright's FormMail, and Thompson Software Products' ObjectAda.

We propose a COA to reduce requirements changes caused by the turnover of decision-makers that is based on the use of specialized requirements engineering teams composed of active duty officers by the geographic and functional Commanders in Chief. These teams use the distributed requirements engineering environment described above to assist in the rapid elicitation of requirements and to increase user participation in the requirements engineering process.

14. SUBJECT TERMS Requirements Engineering, Requirements Analysis, Requirements Management.			15. NUMBER OF PAGES 93
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

FRONT LOADED ACCURATE REQUIREMENTS ENGINEERING (FLARE); A REQUIREMENTS ANALYSIS CONCEPT FOR THE 21ST CENTURY

Anthony E. Leonard
Captain, United States Army
B.S.B.A., Glenville State College, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1997

Author: Anthony E. Leonard
Anthony E. Leonard

Approved by: [illegible]
Luqi, Thesis Co-Advisor

Valdis Berzins
Valdis Berzins, Thesis Co-Advisor

Michael J. Holden
Michael J. Holden, Second Reader

Ted Lewis
Ted Lewis, Chair, Department of Computer Science

ABSTRACT

This thesis focuses on ways to apply requirements engineering techniques and methods during the development and evolution of DoD software systems in an effort to reduce changes to system requirements. The major goal of this thesis is to provide a feasible course of action (COA) that reduces changes to requirements caused by the turnover of DoD decision-makers.

We demonstrate a distributed requirements engineering environment using computer aided software engineering tools linked together with electronic mail. We create this distributed requirements engineering environment using Netscape Communicator, Microsoft's Internet Explorer, Microsoft's Access97 database, Rational Corporation's Rational Rose, Matt Wright's FormMail, and Thompson Software Products' ObjectAda.

We propose a COA to reduce requirements changes caused by the turnover of decision-makers that is based on the use of specialized requirements engineering teams composed of active duty officers by the geographic and functional Commanders in Chief. These teams use the distributed requirements engineering environment described above to assist in the rapid elicitation of requirements and to increase user participation in the requirements engineering process.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

1. The Problem

The high turnover rate of Department of Defense (DoD) commanders results in changing system requirements. New requirements are generated by the introduction of new leaders in positions having the authority to influence the development of hardware and software systems. It is common practice to rotate high level decision-makers out of their positions every two years or less. Each new decision-maker has "their way" of conducting business, and they often make significant changes in their organization's working procedures. This dynamic environment, caused by decision-maker turnover, makes it difficult for any software program to remain consistent throughout its life cycle. For obvious reasons, this practice also is a major contributor to changing requirements; users' needs change with changes in users.

The United States wasted an "estimated 100 billion dollars in 1996" on failed software systems [Ref. 1:p. 73]. The DoD is responsible for a significant portion of this, and with today's shrinking budget, the DoD needs to eliminate the number of undelivered, unusable, and unwanted software systems. Additional domain-expert involvement in the requirements engineering process is needed to achieve this reduction. This thesis addresses this problem by proposing a process that is based on the use of specialized requirements engineering teams by commanders of unified commands [Ref. 2:p. IV-5] to manage software system requirements enabling the DoD to purchase software products that meet their needs, by designing a tool to support this process, and by assessing the effectiveness of this new tool.

2. Requirements Engineering Defined

This thesis addresses two fundamental problems associated with Requirements Engineering: "Problems of investigating the goals, functions, and constraints of a software system; [and] Overcoming barriers to communication…" [Ref. 3:p. 215]. We use the following definition of Requirements Engineering. "Requirements engineering is the disciplined application of scientific principles and techniques for developing, communicating, and managing requirements" [Ref. 4:p. 68].

3. Requirements Engineering Challenges Unique to DoD

Accurate determination of requirements is essential to the defense community's efforts to eliminate unwanted, unneeded, or ineffective software systems. Requirements engineering in the DoD is unique because of the complex qualities of the military environment and its dissimilarity with the normal civilian environment. Software developers who are unfamiliar with the military environment are usually uncertain about the exact needs of military users, yet they are expected to accurately determine requirements of software systems that must meet these needs. Additionally, software developers with proven records of accomplishment working with the DoD are not experts in the problem domain. They experience employee turnover, focus only on areas they deem profitable, and their experience with the problem domain is limited to past and current contracts covering specialized stove-piped systems. Research shows that under these conditions, of uncertainty about the problem domain, it is beneficial to have user participation in the requirements analysis phase of the software development life cycle [Ref. 5].

DoD decision-makers are the engines that produce requirements. These new or changing requirements are a primary stimulus that causes entrance into the requirements engineering phase of the software life cycle. This is illustrated in Figure 1. DoD decision-makers are responsible for two events

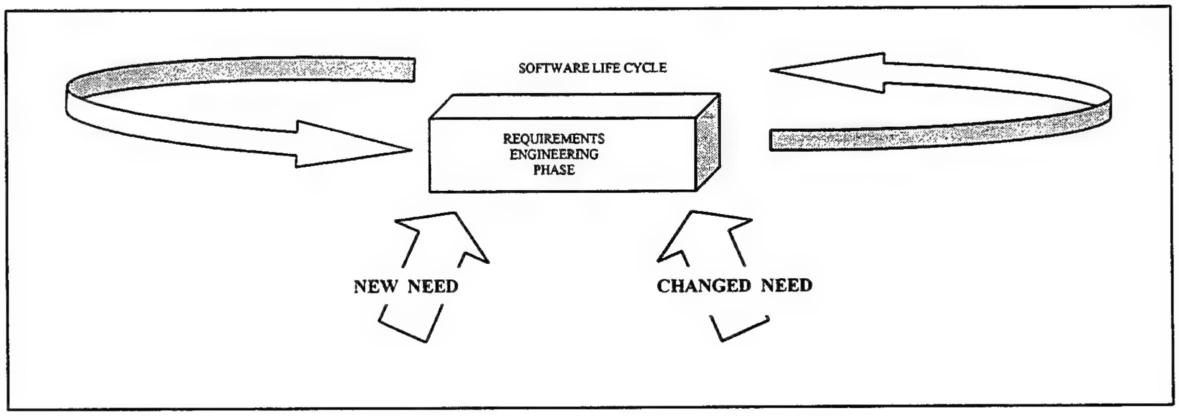


Figure 1. Events Causing Transitions to Requirements Engineering Phase.

that trigger a transition to the requirements engineering phase. The first occurs when a commander determines that an existing software system no longer meets the command's needs. The second is when a new need is identified, automation is capable of satisfying it, and a decision is made to develop a new system or modify an existing one to satisfy the need.

Requirements engineering is important to the DoD decision-maker because nearly every DoD software system is potentially life critical. It is conceivable that the failure of seemingly unimportant, non-critical systems could have adverse effects on the ability of our military forces to fight and win our nation's wars. These critical systems are initially defined and subsequently redefined in the requirements engineering phase. Poor application of requirements engineering techniques, principles, and methods can introduce unnecessary risk into the software system development process.

4. Software Developers Lack Problem Domain Expertise

The DoD relies mostly on civilian employees and contractors to analyze, design, implement, test, evolve, and maintain its computer hardware and software systems. The DoD's civilian Software Engineers and contractors may have close associations with uniformed personnel, and undoubtedly some of their employees are veterans, but taken as a whole they lack expertise in the problem domain of the uniformed DoD decision-maker. This is especially true for the least mobile members of the development team, the programmers.

Lack of communication between programmers and problem domain experts complicates coding efforts of the development team. In the context of a software life cycle consisting of "analysis, requirements definition, design, coding, testing, and operation and maintenance [phases]" [Ref.6:p. 397] the programmer is three phases removed from the problem domain. Requirements are vulnerable and care must be taken to preserve the essence of each requirement during the four translations of the user's needs from the problem domain to the coding phase of the life cycle (See Figure 2).

B. SCOPE

We propose a theoretical requirements engineering team made up of active duty officers, and show how this team can use commercial off the shelf (COTS) and government of the shelf (GOTS) software tools to determine, record, and manage user requirements. We show how this Front Loaded Accurate Requirements Engineering (FLARE) Team fits into the acquisition process. Finally, to show that the use of a FLARE Team is feasible, we provide a case study of how Software Engineers used the same

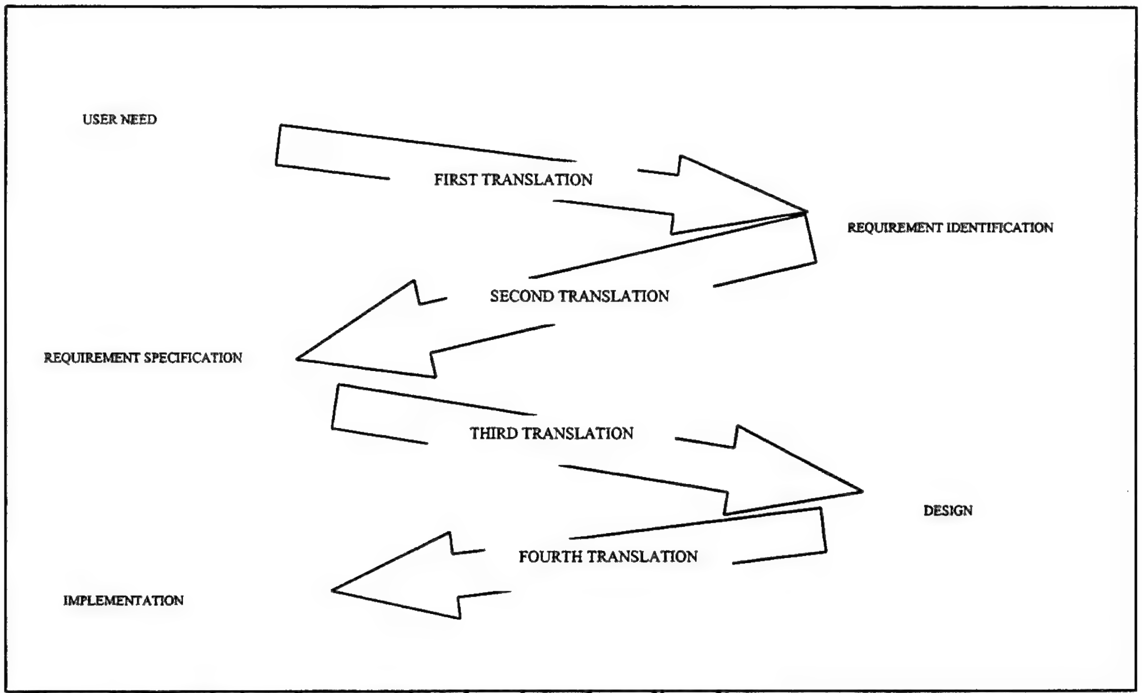


Figure 2. Translations from problem domain to the coding phase.

methods, techniques, and principles that our theoretical FLARE Team would use to produce a new computer-assisted software engineering (CASE) tool appropriately named FLARE.

C. THESIS ORGANIZATION

Chapter I outlines why requirements engineering is important to DoD decision-makers. Chapter II provides an overview of three common requirements engineering paradigms: Object Oriented Analysis (OOA) [Ref. 7, 8, 9], Analysis Using Prototypes (AUP) [Ref. 10], and analysis using Use-Cases [Ref. 11]. A brief overview of the most significant DoD requirements engineering work is provided with an assessment of how the three requirements engineering paradigms fit into the FLARE process. Chapter III details the purpose of FLARE Teams, their composition, and their placement within the DoD software acquisitions system. Chapter IV answers the question: What easy-to-use methods and tools designed to assist Software Engineers in the solicitation, determination, and recording of requirements are available to this hypothetical team? We show how to use Internet technologies to assist in the requirements engineering effort, and we show how the dramatic reduction in information storage costs allows Software Engineers to economically represent the problem domain using audio and video. Chapter V introduces FLARE, a new requirements engineering tool developed for this thesis that will aid in the communication and management of system requirements. Chapter VI contains a case study of the development of a CASE tool using FLARE. Chapter VII discusses topics that warrant further research and summarizes the research contributions made in this thesis.

II. PREVIOUS WORK

A. INTRODUCTION

This chapter provides a brief overview of the major requirements engineering paradigms commonly used to elicit and manage requirements. Additionally, the major requirements engineering work done by the DoD is summarized.

Software Engineers have developed a multitude of methods and tools to aid them in their effort to manage requirements [Ref. 12, 13]. The majority of these methods and tools support Software Engineers as they use one of the three major requirements elicitation paradigms: Object Oriented Analysis [Ref. 8, 9], Analysis Using Prototypes [Ref. 10], and analysis using Use-Cases [Ref. 11].

B. REQUIREMENTS ENGINEERING PARADIGMS

1. Object Oriented Analysis

Of the three common requirements analysis paradigms, Object Oriented Analysis is the most frequently mentioned. Object oriented techniques allow Software Engineers to wrap problem domain concepts into independent entities. These entities can promote reuse and allow engineers to simplify the concepts that are found in the problem domain using abstraction. Once an object is defined, it can be copied and inserted into the software engineering process at any time. Abstraction, inheritance, and polymorphism are the strengths of OOA. [Ref. 7]

OOA's main weakness is its lack of formality. The products produced using OOA, namely the object models, can be interpreted in more than one way. This ambiguity introduces risk into the analysis process because it is impossible to guarantee that various Software Engineers in the development team will have a common interpretation of the object models after they are created. [Ref. 14] This problem can be addressed by formalizing OOA using logic, but that requires advanced training for the analysis team [Ref. 15].

2. Analysis Using Prototypes

Recorded requirements rarely reflect the actual requirements of a software system. Prototypes allow users to discover additional requirements during the developmental phase of a project as opposed to discovering them upon delivery of the system, which happens using the other two requirements engineering paradigms. This iterative requirements elicitation process allows users to modify inappropriate system requirements before delivery. Additionally, the software engineering environments that are provided by prototyping tools provide a means to manage requirements throughout the life cycle of a system. [Ref. 16]

Analysis using prototypes is a highly interactive way to conduct requirements engineering, provided the actual users of the system fully participate. Prototypes are a catalyst for communication. The paradigm is based on showing the user an approximation of the system before the actual system is delivered. The user's reactions are observed, and the set of requirements of the system is modified based on the user's reaction to the prototype. [Ref 17:p. 18]

3. Analysis Using Use-Cases

Software Engineers use this paradigm to elicit requirements from the problem domain through identification of events that occur in the problem domain. The objects linked with these events are identified and recorded as being associated with the event. In addition, the conditions existing before, during, and after each event are recorded. These conditions are used in the design and implementation phases to assist in the development of preconditions, invariants, and post-conditions.

The paradigm was developed by Ivar Jacobson and has been widely adopted by the software engineering community. [Ref. 11] Two of the newest software engineering methods, the OCTOPUS method [Ref. 18] and the Unified Modeling Language [Ref. 19], rely heavily on the Use-Case paradigm to elicit the requirements from the problem domain.

4. Assessment and Recommendation

FLARE teams will use all three paradigms. Each provides unique capabilities to Requirements Engineers. Initially the teams will use OOA to identify objects. The teams will use these objects to conduct an exhaustive Use-Case analysis; updating the object models as new objects are identified. The teams will use prototypes of the proposed system as soon as enough information is elicited from the problem domain to identify an appropriate predefined prototype to present to users. The more experienced members of the teams will use formal methods to described critical portions of the proposed system as they are identified. The teams will use the CASE tool FLARE that we introduce in this thesis to manage the information produced in the requirements development process. Additionally, the team will use the tool to facilitate the communication of the various object models, functional models, use-case diagrams, prototypes, formal requirements specifications, and informal requirements specifications.

C. DOD REQUIREMENTS ENGINEERING ORGANIZATIONS

The user of a software system is responsible for the identification of the requirements of the software systems they purchase, but most users lack the ability to determine and specify their requirements in a meaningful way. However, some components of the DoD have dedicated considerable resources to requirements engineering. We mention three of them in this thesis. The U.S. Defense Information Systems Agency (DISA) uses requirements engineering methods on the Global Command and Control System (GCCS) and the Defense Information Infrastructure Common Operating Environment (DII COE) [Ref. 20], which are two large software systems. The U.S. Naval Research Laboratory (NRL) has adopted the work of David Parnas [Ref. 21] to produce an in-house formal method to specify requirements that has been used on large software projects [Ref. 22]. The U.S. Naval Postgraduate School's (NPS) Computer Science Department is developing a software engineering tool that promises to enhance a Software Engineer's ability to engineer requirements.

1. DISA

DISA's D7 department is devoted to the elicitation and analysis of joint military requirements [Ref. 23]. It appears that DISA is the only organization within the DoD actively using the Internet to aid in the management of requirements. On their Internet home page for the Common Operating Environment (COE) they provide a link to the "Software Requirements Specification (SRS) for the Defense Information Infrastructure (DII) Common Operating Environment (COE) Common Support Applications." [Ref. 24] At this site, DISA has taken the first step of audio/visual representation of requirements by providing, where appropriate, a picture to augment the written requirement.

2. NRL

NRL has developed an informative tutorial covering requirements engineering titled "Software Requirements: A Tutorial" [Ref. 25] that explains why an accurate and sufficient requirements analysis of the problem domain is critical for successful software systems development. The tutorial explains the differences between the major requirements engineering techniques and presents a good explanation of why it is so difficult for users to know and accurately specify what they want from a software system.

NRL has also developed the software cost reduction (SCR) method, a requirements engineering tool that shows promise in reducing the failure rate of software systems. It does this by adding formalism into the requirement engineering process. Specifically, requirements elicited from the problem domain are specified formally, preventing ambiguity of the representations of the requirements. [Ref. 26]

3. NPS

Researchers in the Software Engineering Track of the School's Computer Science Department have developed a CASE tool to aid in the elicitation and management of requirements using prototypes. The tool is called Computer-Aided Prototyping System (CAPS). It promises to reduce the amount of time it takes to develop a system by various means that involve user participation in the development of prototypes. [Ref. 27, 10] The researchers at NPS have significantly increased the software engineering community's understanding of the benefits provided by using prototypes. The following quote succinctly describes the potential benefits provided by the school's research with CAPS:

> Traditional [software life cycle] approaches to software development produce working code only near the end of the process. When utilized during the early stages of the development life cycle, rapid prototyping allows validation of the requirements, specification and initial design before valuable time and effort are expended on implementation software [Ref. 28:p. 77].

D. SUMMARY

This chapter described three primary requirements analysis paradigms, Object Oriented Analysis, Analysis Using Prototypes, and analysis using Use-Cases. FLARE Teams will use all three paradigms in their requirements engineering efforts.

DISA and NRL are two organizations within DoD that have established a requirements engineering capability. DISA is involved in the management of the requirements of major software systems such as GCCS and the DII COE. NRL has developed a formal requirements engineering method, the SCR method, that promises to contribute the community's efforts to reduce failures of software systems by eliminating ambiguity in the specifications of requirements.

The Naval Postgraduate School is developing CAPS. When finished, this tool will enhance the requirements engineering capability of the software engineering community.

III. A REQUIREMENTS STABILIZATION COURSE OF ACTION

A. INTRODUCTION

This chapter recommends a feasible course of action that would reduce changes to requirements caused by the turnover of DoD decision-makers.

The crux of this course of action is the formation of permanent requirements engineering teams composed of military officers by each of the geographic and functional commanders in chief (CINCs). These teams will conduct requirements engineering for all software projects in the CINCs' command. They also will provide command, control, communications, and computer systems (C4) advice to the commanders [Ref. 29].

Involvement of the Front Loaded Accurate Requirements Engineering (FLARE) Team in the development of a C4 system begins as soon as a commander identifies a need that is potentially solvable by automation. The team will conduct requirements engineering within the command, and it will engineer interoperability requirements.

The Team will make extensive use of Internet technologies, prototyping tools, and computer-assisted software engineering tools, some of which are describe in Chapters IV and V.

B. WHY A FLARE TEAM IS NEEDED

In the DoD, it takes several years to develop and acquire a software system [Ref. 30:p. 70]. If we assume that most organizations change commanders every two years, we find that the commander who identified the need to develop a C4 system will rotate out of his or her position before the first version is delivered for evaluation.

This turnover is a primary reason why requirements change. Each new commander may apply existing doctrine differently to accomplish the command's mission. These differences in operational techniques cause requirements to change. Although the need of a software system is articulated in a "Missions Needs Statement" and an operational requirements document is prepared [Ref. 30], successors of the originator of a software system still may lack complete knowledge and specific insight as to why a system was developed. Hence, a new commander may have a difficult time assessing the effectiveness of delivered software systems that were initiated by his or her predecessors.

Traditionally, the DoD relies on a command's staff to overcome the transitional problem described above. However, two factors make it difficult for staffs to provide new commanders with adequate advice in the area of extended software system development. First, staffs are also affected by high turnover, which erodes institutional knowledge. Second, staffs may lose focus on developmental software systems because the workload associated with meeting the requirements of day to day operations leaves little time to manage the requirements of developmental software systems.

The acquisition community is addressing some of these problems by adopting new acquisition methods and by reducing the time it takes to deliver products to system stakeholders; people who interact with or are affected by a system. The DoD's use of Integrated Product Teams in the acquisition process is a step in the right direction because the user is represented at essential decision meetings.

The trend in the DoD towards streamlined acquisition and the use of best industry practices requires a fundamental change in the way we conduct requirements engineering. Our leaders "must get personally involved in understanding the relative costs, benefits, risks, and returns associated with information technology investments they are making decisions about". [Ref. 31] FLARE Teams will facilitate this.

C. FLARE TEAM MISSION AND COMPOSITION

We propose the DoD take further actions than those outlined above and use FLARE Teams to accomplish the requirements engineering tasks associated with the development of software intensive systems. This team would support CINCs and system stakeholders by compressing the requirements engineering phase, by providing institutional knowledge of developmental software systems, and by conducting requirements engineering for the command. This team is the embodiment of the three reasons why software systems succeed: "user [stakeholder] involvement, executive management [CINCs] support, and a clear statement of requirements" [Ref. 32].

1. Team Mission

The FLARE Team will perform all requirements engineering functions for the command; provide advice to the commander on C4 system issues; and represent the command in the "Integrated Product and Process Development System [(IPPD)]" [Ref. 33].

2. Team Composition

The team will consist of eight officers. This quantity is consistent with traditional team sizes used in DoD. When the number of automated systems within a particular command is too large for a team of this size to effectively manage the command should either recursively form additional teams to satisfy requirements engineering demands or temporarily increase the number of members on the team to meet the excessive demands.

An officer with the rank equivalent to a Lieutenant Colonel will lead the team. An officer of this rank possesses significant knowledge of the problem domain and has developed leadership skills that will allow the officer to effectively lead and mentor the other team members. Additionally, once this course of action is institutionalized within the DoD, an officer of this rank will have served on a FLARE Team as a Major or Lieutenant Commander, which likely would enhance the officer's ability to lead the FLARE Team.

Majors and Navy Lieutenant Commanders will fill the other seven positions. Officers of this rank have worked in various areas of the problem domain for several years and have attended their respective service's developmental schools. The possession of these two qualities, experience and formal military education, allows them to understand problem domain concepts more easily than civilian contractors.

Each member must have an advanced degree in software engineering. In general, the lack of such formal software engineering education would prevent team members from effectively conducting requirements engineering. Requirements engineering is a complex activity requiring the application of scientific principles to the elicitation, communication, and management of requirements [Ref. 4]. The possession of an advanced degree would enable team members to apply the needed scientific principles.

D. METHODS USED BY THE FLARE TEAM

The Team will use a combination of the traditional requirements engineering paradigms described in Chapter II of this thesis. The team will use a requirements engineering tool (FLARE) designed specifically for the proposed FLARE Team's use to communicate and manage the requirements of a command's software systems. A thorough description of this tool is provided in Chapter V of this thesis.

E. FLARE TEAM'S ROLE IN THE SOFTWARE ACQUISITION SYSTEM

1. Team's Interaction with the Problem Domain

The first stage of requirements engineering is the elicitation of requirements from the problem domain. This is a primary function of the FLARE Team. The Team begins this process as soon as a CINC identifies a need to develop a software system. The FLARE Team will write the Mission Needs Statement (MNS) that formally identifies an existing or future deficiency within the command that may affect the command's ability to accomplish its mission [Ref. 34]. It will present a briefing to the commander detailing the MNS and providing an initial assessment of the systems, equipment, and personnel needed to meet the new need. While preparing this briefing, the team will explore the possibility of satisfying the need using existing GOTS systems. Throughout the entire process the team must record and place requirements in an understandable format. The Team will validate these requirements with all stakeholders. The team will record which constituency supports each requirement and the consequences to be expected if the requirement is not met. This will assist the CINCs in making costs versus benefits decisions.

CINCs are members of the problem domain and are responsible for deciding which software systems their commands will purchase using their commands' operational funds. Additionally, they can influence what Acquisition Category I, II, and III programs [Ref. 34] are approved. When appropriate, FLARE Teams will provide information briefings to their CINCs detailing the advantages and disadvantages of the software systems under procurement consideration at the Defense Department level, which would enable CINCs to make more informed recommendations to the Defense Acquisition Board [Ref. 35].

2. Team's Interaction with Program Managers (PM)

This course of action maintains the current focal point for the acquisition of software systems: the PM. FLARE Teams will work closely with PMs. For new projects, the Team will provide the PM with the initial database of requirements that the team has elicited from the problem domain. The Team will also give the PM the preliminary operational requirements document that they have produced.

Programs that affect multiple geographic and functional CINCs will require additional coordination on the part of the PM because each FLARE Team will have elicited a unique set of requirements from their respective commands. This may appear to be a duplication of effort, but it is not. Each CINC has a unique mission; therefore, it is likely that each FLARE team will produce a different set of requirements. The PM need only take the union of all the sets of requirements produced by the different FLARE Teams to identify the total requirements of all the commands affected by the new program. The intersection of all the sets of requirements will give the PM an indication of the most important requirements, assuming that each CINC's requirements are of equal importance.

Typically, program managers' main concern is the development and acquisition of systems on time and within budget. FLARE Teams will play a crucial role in helping program managers achieve this by acting as a requirements stabilization mechanism for the project. FLARE Teams will brief new CINCs on the status of each software system under development that will affect their command and the logic behind the previous CINC's decision to support their development.

FLARE Teams will represent their commands in all matters concerning software requirements. When a cost versus benefits decision must be made the FLARE Team will give the CINC a detailed briefing describing the situation. [Ref. 36]

3. Team's Interaction with Integrated Product Teams (IPT)

Time and personnel permitting, each IPT [Ref. 33] will have a FLARE Team member on it that will ensure issues concerning requirements are recorded and addressed. When the number of IPTs formed to address software system acquisition issues becomes too large for the Team to place a representative on each the Team will provide representatives to the most important IPTs. We do not mean to imply that we view the IPT process as unimportant; rather, we simply recognize the fact that it may be impossible to provide full representation.

4. Team's Interaction with Developers

The FLARE Team will coordinate all developer activities within the command. Develops' requirements questions will be answered by the Team, and the Team will ensure requirements are satisfied by delivered systems.

During the maintenance and evolution phases of a software system, the FLARE Team will continue to manage requirements and assess the amount of progress made by maintainers and evolvers.

F. SUMMARY

This Chapter presented a course of action designed to stabilize requirements of software systems developed by CINCs, which will reduce the number of failed software systems in the DoD. The course of action relies on the formation of permanent requirements engineering teams consisting of eight officers, all possessing advanced degrees in software engineering, and lead by an O5. Each CINC will have one or more FLARE Teams depending on the demand. The team's mission is:

> The FLARE Team will perform all requirements engineering functions for the command, provide advice to the commander on C4 system issues, and represent the command in the IPPD System.

The integration of this team, as the decision-maker's representative, into the acquisition process would enhance the effectiveness of the entire software development process by giving the decision-maker the ability to make informed decisions and by providing a stable source of information for developers.

IV. BRINGING THE PROBLEM DOMAIN TO THE IMPLEMENTATION DOMAIN

A. INTRODUCTION

This Chapter shows how FLARE Teams can use Internet technologies to enhance the effectiveness of the set of CASE tools [Ref. 37] used by the teams to manage requirements. They must ensure that each requirement is satisfied in the implementation domain by an automated system [Ref. 15:p. 34]. Ideally, the set of requirements should be managed throughout the evolution of the system to provide the rationale for the system's behavior [Ref. 38]. We show how to extend formal and informal specifications with audio and video file representations of requirements [Ref. 39, 40]. This is valuable because video allows developers to quickly gain a conceptual understanding of specifications, breaks the mind numbing monotony often experienced when reading formal textual specifications and graphical diagrams, and effectively provides an abstract representation of objects found within specifications.

Additionally, we demonstrate how Internet technologies can help managers improve their task assignment methods by incorporating programmers' assessments of the difficulty of implementing software components into the decision process. The products produced by the FLARE Teams are used to make this possible.

B. CASE ENVIRONMENT ENHANCEMENT USING INTERNET TECHNOLOGIES

The number of computer aided software engineering tools and environments available to assist FLARE Teams is extensive. Queen's University in Kingston, Ontario publishes a partial CASE tool list that has nearly 400 tools listed [Ref. 41]. We use a small subset of available CASE tools, those commonly used at the Naval Postgraduate School, to illustrate how we can enhance a CASE environment with Internet technologies.

Researchers at the Naval Postgraduate School have developed a CASE tool called Computer-aided Prototyping System (CAPS) [Ref. 10]. This tool provides a capability to develop prototypes using the prototype system description language (PSDL) [Ref. 42]. Once completed, CAPS promises to provide a robust environment that will facilitate the management of requirements throughout the life of a system.

By design, the prototypes produced using CAPS are demonstrated to users. Users evaluate the prototypes, and developers use the information obtained from the user's evaluation to refine the requirements of the software system [Ref. 43]. Intuitively, it seems that CAPS would produce the best results if a developer personally presented a prototype to a user, but this would be expensive in terms of travel and set up time, especially if multiple meetings between a developer and user were needed. By using audio and video conferencing techniques over the Internet, similar to those described by Macedonia and Brutzman [Ref. 44], we can remove this limitation. Additionally, the developer's ability to interact with the user at any time, provided the user has access to an Internet enabled computer with video conferencing capabilities, likely would enhance the engineering environment created by CAPS and tools similar to it. This enhancement would be achieved by allowing the developer to resolve ambiguous requirements with users while they are still actively involved in the process of developing a prototype or model. It also would reduce the cost of travel by eliminating the requirement of having the developers and users co-located during the presentation of a new or changed prototype. Applications exist on the market that make this possible with a modest, under $1,000.00, investment in additional equipment and software [Ref. 45, 46].

The use of Internet video conferencing to augment a software-engineering environment is available today; as are other Internet technologies that provide comparable enhancements. One of these additional Internet technologies is "intelligent browsing" [Ref. 47]. It is now possible for a Software Engineer to use intelligent agents to retrieve information from the Internet [Ref. 48]. These tools can aid Software Engineers in their efforts to understand the problem domain and to find appropriate solutions to requirements in the implementation domain. It seems that a software engineering environment enhanced with intelligent agents and Internet video conferencing would significantly increase a Software Engineer's ability to produce quality software in a timely manner.

C. AUGMENTATION OF FORMAL AND INFORMAL SPECIFICATIONS WITH VIDEO

Where appropriate, FLARE Teams will augment requirement specifications with video representations of the requirements. This would be helpful because certain requirements can be better understood by developers if they can watch users during the performance of the activities that generate the requirements [Ref. 39]. For example, most software developers are not experts of infantry fighting procedures and have no concept of the actual tactics, techniques, and procedures used by infantry forces to accomplish their assigned mission. This lack of understanding of the problem domain is further complicated by preconceived ideas formulated by software developers as they are exposed to the entertainment industry's dramatization of infantry soldiers and their fighting techniques. Problem domain objects and concepts such as foxhole, field of fire, accurate, timely, and cover have very specific meanings to infantry soldiers. Typical Software Engineers do not necessarily share these meanings. Software Engineers can represent each of them with formal or informal methods. However, would a Software Engineer located in Silicon Valley, when given a formal or informal representation of the requirements elicited from this problem domain, be able to design a system that would satisfy the needs of the user without direct knowledge of the user's context? The requirements produced from this type of problem domain, a domain that is foreign to most Software Engineers, is an ideal situation to use video to augment requirements. In this section, we show how the addition of a type "video", similar to that of a textual comment, to formal and informal specification and design languages will increase developers understanding of the problem domain.

1. The New Memory Paradigm

In the use of video to augment the representation of requirements, we would like to have easy and quick access to it. The cost of existing magnetic storage has recently dropped to affordable levels, making storage of video on fast hard disk drives feasible. Figure 3 depicts the dramatic reduction in memory prices that have taken place during the ten-year period between 1987 and 1997. It is now economically feasible to augment requirement specifications with video that is retrievable by anyone in the software development group on demand. This capability is the crux of bringing the application domain to the design and implementation domains. Another storage technology, digital versatile disk (DVD), introduced to the masses in 1997 provides additional space to store audio files [Ref 49].

The same technology that allows analyzing "a golf swing from up to nine different camera angles" [Ref. 50] can be used to provide on-demand video to increase software and systems developers' understanding of the problem domain. Adding a type "Video" to specification and design languages provides an easy way to incorporate the use of video into the software engineering process.

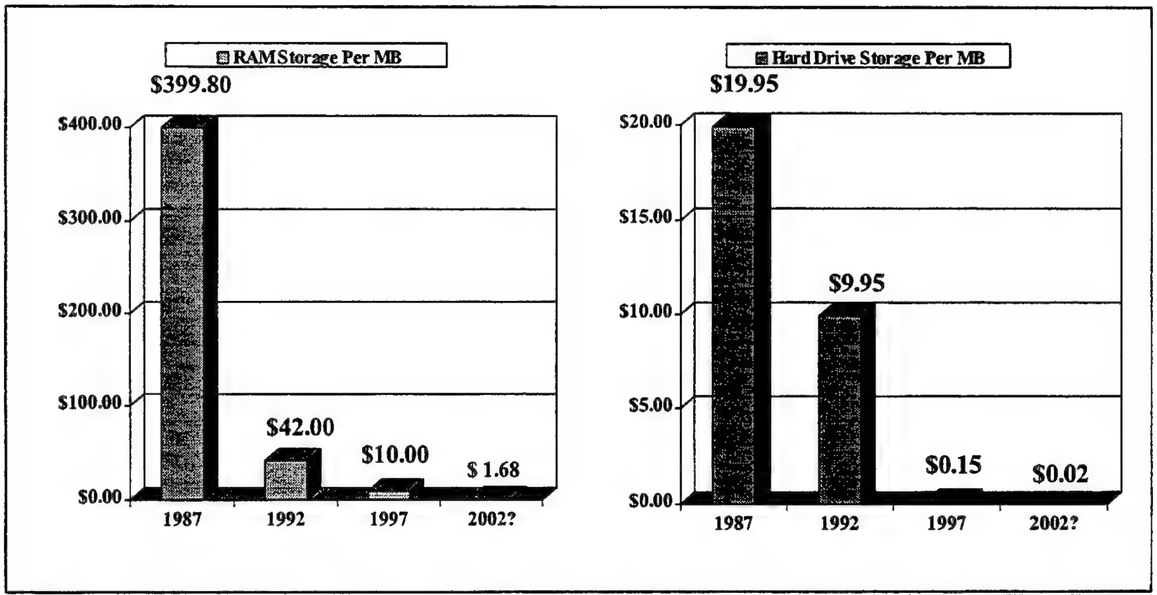


Figure 3. Dropping Memory Prices. In 1987, the Cost of Secondary Magnetic Storage, Hard Drives, was about $20.00 a Megabyte (MB) [Ref. 51:p. 89], and Primary Memory, Random Access Memory (RAM), was about $400.00 a MB [Ref. 52:p. 309]. In January of 1992 this Dropped to about $10.00 a MB for Hard Drive Storage and $42.00 a MB for RAM [Ref. 53:p. 356]. In January of 1997, both types of Memory were at an all-time low. Hard Drive Storage Sold for about $.15 a MB [Ref. 54:p. 152], and RAM for about $10.00 a MB [Ref. 55:p. 153]. The prices shown for the year 2002 are based on a 30% yearly reduction in memory prices [Ref. 56]

2. Augmentation of Specification and Design Languages with Video

We use the Spec Language [Ref. 15] to illustrate how FLARE Teams would incorporate video into the development process. There are two ways to use video with Spec. Both require that the Spec model be saved in HTML format [Ref.57] and the use of an Internet browser to access the model.

The first way to utilize video is to include a comment anywhere in a Spec model stating that a video clip is available; hyperlinking this comment to the video clip. This is attractive because it allows the designer to quickly augment a model with the ease of using comments. Figure 4 shows how to implement

```
DEFINITION bunker -- Concept for describing shelters. - - Click here to view video clip
  INHERIT fortification - - The module "fortification" defines types security and cover.
.
.
END
```

Figure 4. We Have Added the Bold Text on the First Line. This Comment Is Linked to A Video File Describing a Bunker.

this method. The other way to incorporate video is to define a new concept "hyperlink" and create instances of type hyperlink when appropriate. Figure 5 shows the definition of concept hyperlink. Any instance of this concept would be a hyperlink to some type of file, video in this case.

```
DEFINITION link  - - Concepts for describing hyperlinks.

  CONCEPT link: type - - The set of hyperlinks.

END
```

Figure 5. Definition of Concept Link.

D. PROGRAMMER INPUT INTO THE WORK TASKING PROCESS

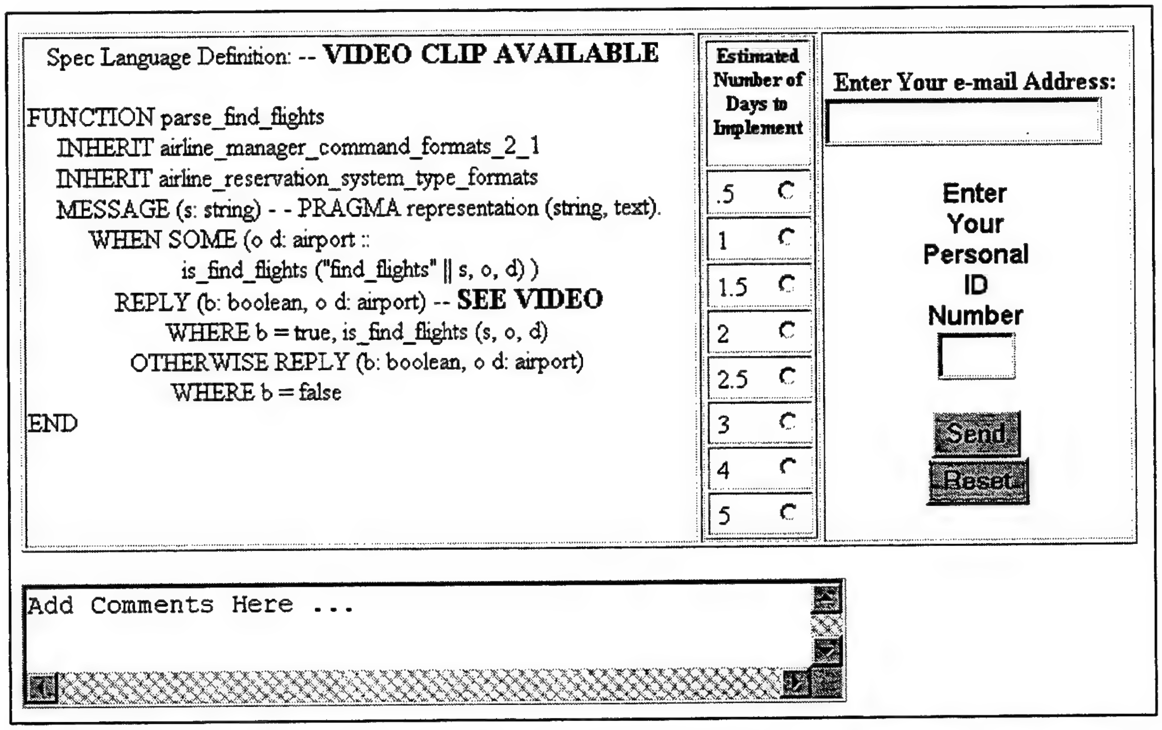
Once designers have identified modules that require implementation, management must produce a programmer work schedule [Ref. 58]. The products produced by FLARE Teams enable programmers' to gain a better understanding of problem domain concepts and objects. This increased understanding makes programmers' input into the module assignment process used by managers more valuable.

Each programmer knows their programming abilities and can estimate the time required to complete a programming task. We show how managers can assign tasks to programmers based on these estimates. To do this, we use Internet technologies to produce an interactive module evaluation environment where each uncommitted programmer rates every unassigned module by perceived level of difficulty.

This process allows managers to produce an optimal programmer work schedule. An optimal programmer work schedule is a work schedule designed to minimize the cost of implementing all unassigned modules. Cost is measured in terms of time, where a shorter implementation time is better.

We have developed an Internet form [Ref. 59] that uses a common gateway interface (CGI) script, FormMail [Ref. 60], to capture programmers' assessments of tasks (Figure 6). The method used to gather input requires that each programmer estimate the number of days it would take to implement the module listed on the form. Each programmer is required to repeat the process until they meet one of the following three criteria. They have identified a module that they can implement in minimal time, and the system schedules the programmer to implement it; the programmer has evaluated all modules that have not been scheduled; or management stops the process because they have determined a suitable working schedule.

Once the Send button is pressed, the input provided is automatically emailed to a central location



Spec Language Definition: -- **VIDEO CLIP AVAILABLE**

```
FUNCTION parse_find_flights
   INHERIT airline_manager_command_formats_2_1
   INHERIT airline_reservation_system_type_formats
   MESSAGE (s: string) - - PRAGMA representation (string, text).
       WHEN SOME (o d: airport ::
                  is_find_flights ("find_flights" || s, o, d) )
         REPLY (b: boolean, o d: airport) -- SEE VIDEO
               WHERE b = true, is_find_flights (s, o, d)
           OTHERWISE REPLY (b: boolean, o d: airport)
                WHERE b = false
END
```

Estimated Number of Days to Implement
.5 ○
1 ○
1.5 ○
2 ○
2.5 ○
3 ○
4 ○
5 ○

Figure 6. Form Used to Capture Programmers' Assessment of the Time Needed to Implement a Module. The Spec Language Definition Was Developed by V. Berzins [Ref. 15:p. 424].

where it is processed. Ideally, the scheduling process will be automated using techniques similar to those of the Evolution Control System (ECS) [Ref. 61] with the addition of incorporating programmer input into the system. The modified ECS can enforce various policies ranging from scheduling a task immediately if a programmer estimates they can complete it in .5 days, to waiting until each programmer has evaluated all modules in an attempt to develop an optimal schedule.

This system can be used to assess schedule risk. Modules with a wide variance in programmer estimates are more likely to cause problems than the modules where most of the programmers agree on the time it would take to implement.

This system gives managers the ability to collect additional information on their programmers. For example, programmers' who continuously have a large variance between estimates and actual implementation time may require additional training on understanding specifications.

Incorporating programmers' assessments into the process also allows management to assign tasks to programmers in a way that takes advantage of each programmer's personal knowledge base. That is, each programmer has a class of problems that they can easily solve due to their accumulated experiences and habits. Incorporating their input into the module scheduling process seeming would increase their productivity because they would be assigned modules based on their completion time estimates. Additionally, this system can be utilized to focus recruiting efforts. Consider a situation in which the entire set of programmers rated tasks C, D, and E as taking the maximum allowable time to implement. Management could focus their recruiting efforts on finding individuals that rate these tasks as taking minimal time to implement, thereby increasing the efficiency of the entire organization and minimizing costs.

E. SUMMARY

FLARE Teams can enhance the understanding of the problem domain by using existing Internet technologies. These technologies also provide a means to enhance CASE environments. Internet video conferencing, on-demand video, intelligent agents, and CGI scripts are some of the technologies that are easily incorporated into a CASE environment.

Inexpensive memory and information storage makes it affordable to augment requirement specifications with video representations of the problem domain. This enhances the entire development process by providing another way to represent problem domain concepts and objects.

The use of Internet technologies by FLARE Teams and developers enhances the ability of management to incorporate programmers' assessments of the difficulty of completing tasks into the scheduling process.

V. FLARE: A REQUIREMENTS ENGINEERING ENVIRONMENT

A. INTRODUCTION

In the previous Chapter, we stated that traditional software engineering environments could be easily enhanced using Internet technologies. We offer proof of this by introducing a CASE tool called FLARE that uses the technologies presented in Chapter IV to create a distributed requirements engineering environment. FLARE is designed to enhance the software development process by offering a means to inexpensively manage requirements and facilitate communication of requirement related issues between all interested parties in the software development process. The FLARE Team discussed in Chapter III would use this tool.

B. FLARE'S COMPOSITION

FLARE is composed of several software programs that are coupled by electronic Internet mail. This coupling produces a synergistic effect by combining the distinct features of each program to produce a requirements engineering environment. The following programs create the FLARE environment. Each contributes unique properties.

1. Microsoft's Access 97

This is an inexpensive database designed to function on Windows 95 or Windows NT. This database makes it relatively easy to manipulate the requirements engineering information entered into the FLARE environment. It also produces reports in HTML format. This enables users of FLARE to easily publish information that has been manipulated by database methods to the Internet. [Ref. 62]

2. An Access Database File; FLARE.mdb

This database file contains the tables, queries, forms, reports, and macros that constitute the management aspects of FLARE. [Ref. 63]

3. An Electronic Mail File Parser; FLARE.exe

We developed this electronic mail file parser to extract only FLARE related information from an electronic mail file. We wrote the parser in Ada 95. Appendix A contains the source code listing.

4. A Set of JavaScript Enhanced HTML Files

This set of files, when accessed with an Internet browser, creates the user interface for the FLARE environment. The essential elements of these files are embedded JavaScript [Ref. 64] and Forms [Ref. 65]. JavaScript enables the pull-down menus found in the user interface to function. The ability to input and transmit information is made possible by using Forms embedded in FLARE's HTML files. The source code for the files is given in Appendix B.

5. A JavaScript Enabled Internet Browser

The browser is the shell that the user interface of FLARE runs in. It must be JavaScript enabled to allow the pull-down menus to operate. We used Microsoft's Internet Explorer [Ref. 62] and Netscape's Communicator [Ref. 66] to test the user interface.

6. An Electronic Mail Program

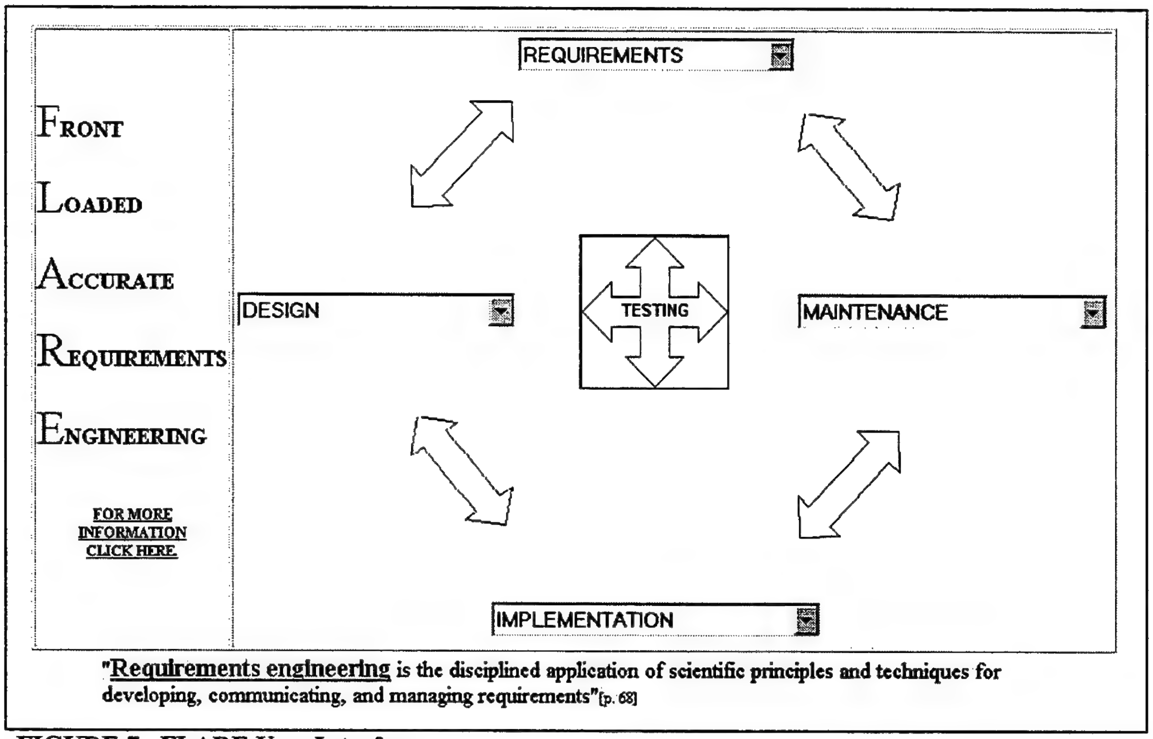
This is the mechanism we use to implement the distributed properties of FLARE. Information is submitted to a central location via electronic mail where it will eventually be converted into a format usable by the database by the parser we described above. We used the electronic mail programs provided with Microsoft's Internet Explorer and Netscape's Communicator.

7. FormMail

This CGI script running on NPS's Internet server was used to pre-format information submitted by users using FLARE'S user interface. FormMail makes information entered into HTML Forms human readable. It allowed the rapid development of the electronic mail parser described above.

C. FLARE'S USER INTERFACE

Figure 7 shows the initial user interface of FLARE. Each of the four pull-down menus represents a phase in the software life cycle. Each menu shares the "Mission Needs Statement" option. We chose to include this in each phase to emphasize the needs of the customer. The arrow between the "REQUIREMENTS" and "DESIGN" menus symbolizes communication between the two phases in the form of requirements specifications. The arrow between the "DESIGN" and "IMPLEMENTATION" phases symbolizes communication between the phases in the form of a formal or informal design specification. The arrow between the "MAINTENANCE" and "REQUIREMENTS" phases symbolizes



"Requirements engineering is the disciplined application of scientific principles and techniques for developing, communicating, and managing requirements"[p. 68]

FIGURE 7. FLARE User Interface.

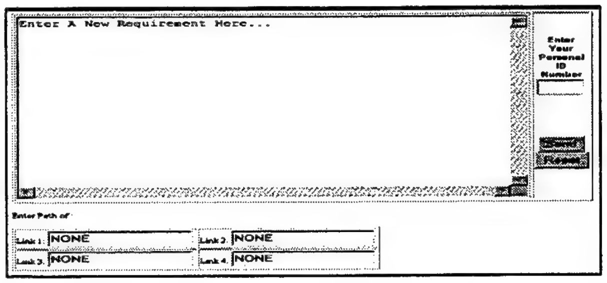
the transition caused by new or changing requirements. The "TESTING" icon in the center of the interface with arrows radiating in the four directions symbolizes the testing that must be built into each phase of the development cycle.

We describe the functionality of each menu option in the following subsections.

1. Requirements Pull-Down Menu

a. *Enter Requirement*

This option allows a Software Engineer to input a new requirement into the FLARE environment. Figure 8 shows the format of the form. The four link fields at the bottom of the form are provided to allow FLARE Team members to include logical links to video or other file representations of requirements. For example, if an engineer entered a requirement that contained the problem domain object foxhole, and the engineer had a 30 second video clip of a foxhole, then the engineer could add a comment in the requirement indicating that a video file of a foxhole was available at link 1. We choose to label these fields 'links" because the FLARE Team could use file types other than video to augment the requirements.


Enter A New Requirement Here...

Enter Your Personal ID Number

Enter Path of

Link 1:	NONE	Link 2:	NONE
Link 3:	NONE	Link 4:	NONE

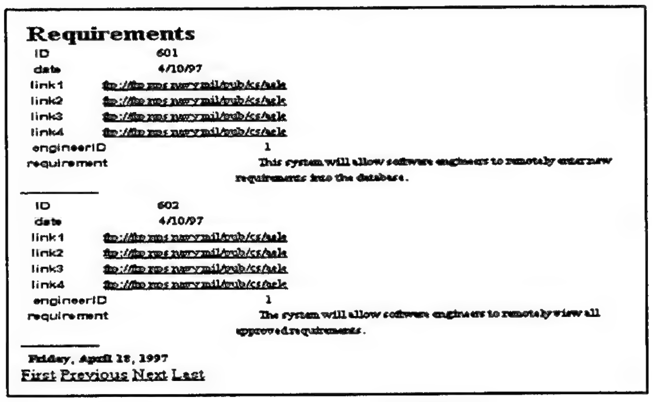
FIGURE 8. Requirements Entry Form.

b. ***View Requirements***

This option allows engineers to view all approved requirements. Figure 9 shows the HTML page that the database generates automatically when given the "Save as HTML" command found on the menu bar of the Access database.

c. ***Ask a Req. Question.***

This option allows a way to input questions about requirements into the FLARE system. It is similar in appearance to the form in Figure 8. Upon execution of the FLARE database program, the


Requirements

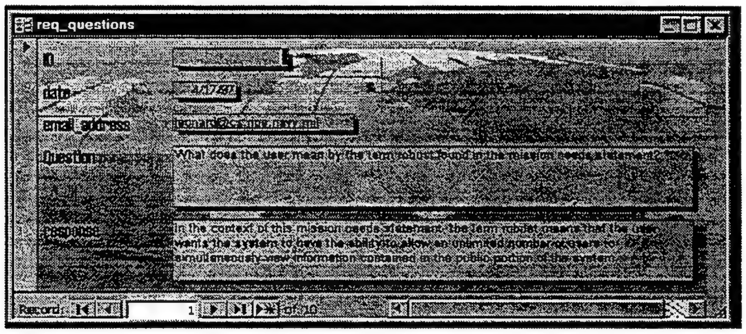
ID	601
date	4/10/97
link1	ftp://ftp.nps.navy.mil/pub/cs/sele
link2	ftp://ftp.nps.navy.mil/pub/cs/sele
link3	ftp://ftp.nps.navy.mil/pub/cs/sele
link4	ftp://ftp.nps.navy.mil/pub/cs/sele
engineerID	1
requirement	This system will allow software engineers to remotely enter new requirements into the database.

ID	602
date	4/10/97
link1	ftp://ftp.nps.navy.mil/pub/cs/sele
link2	ftp://ftp.nps.navy.mil/pub/cs/sele
link3	ftp://ftp.nps.navy.mil/pub/cs/sele
link4	ftp://ftp.nps.navy.mil/pub/cs/sele
engineerID	1
requirement	The system will allow software engineers to remotely view all approved requirements.

Friday, April 18, 1997

Figure 9. Database Generated HTML Page.

question is automatically imported into the database where a FLARE Team member using the form shown in Figure 10 can answer it.


req_questions

ID

date 4/17/97

email_address [illegible]

Question What does the user mean by the term robust found in the mission needs statement?

response In the context of this mission needs statement, the term robust means that the user wants the system to have the ability to allow an unlimited number of users to simultaneously view information contained in the public portion of the system.

Record: 1 of 10

Figure 10. Question Response Form.

d. ***View Requirements Questions***

This option allows users to view questions that have been asked along with the answers provided by engineers using the form shown in Figure 10.

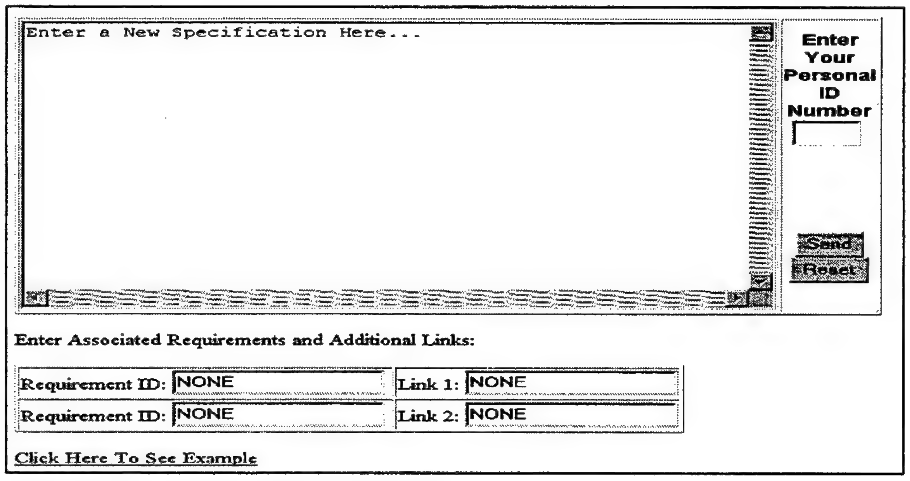
e. ***Mission Needs Statement***

This option allows engineers to review the mission needs statement that prompted the development of the software system.

2. **Design Pull-Down Menu**

a. ***Enter Specification***

This option allows a Software Engineer to input a specification that satisfies a requirement. Figure 11 shows the format of the form used. Note the fields labeled "Requirement ID." These fields facilitate requirements management. When a specification is entered into the system, the engineer should also enter the requirements that are associated with the specification. The link fields on the form enable engineers to enter informal design specifications into the FLARE environment. An engineer would enter a comment in the text area of the Specification Entry Form indicating that a hyperlink to a graphical model or specification exists. An informal specification would likely be in the form of a graphical model such as those found in the Unified Modeling Language [Ref. 19].


Enter a New Specification Here...
Enter Your Personal ID Number
Send
Reset

Enter Associated Requirements and Additional Links:

Requirement ID:	NONE	Link 1:	NONE
Requirement ID:	NONE	Link 2:	NONE

Click Here To See Example

Figure 11. Specification Entry Form.

b. *Remaining Menu Options*

The menu options View Specifications, Ask a Specification Question, View Specification Questions, and Mission Needs statement are very similar to those found in subsection C-1 above and do not require further explanation.

3. Implementation Pull-Down Menu

a. *Enter Estimates*

The functionality of this option is thoroughly described in Chapter IV, Section D.

b. *Remaining Menu Options*

The menu options View Specifications, View Requirements, Ask an Implementation Question, View Implementation Questions, and Mission Needs statement are very similar to those found in Subsection C-1 above and do not require further explanation.

4. Maintenance Pull-Down Menu

a. *Enter a Bug Report*

This option allows a Software Engineer to input an error found in the implementation into the flare system.

b. *Enter Change Request*

This option allows a Software Engineer to input a new or changed requirement into the FLARE environment.

c. *Remaining Menu Options*

The menu options View Bug Reports, View Change Requests, Ask a Maintenance Question, View Maintenance Questions, and Mission Needs statement are very similar to those found in subsection C-1 above and do not require further explanation.

D. ELECTRONIC MAIL PARSER

We developed a parser that identifies a FormMail formatted message randomly placed in a text based electronic mail file. The parser places pertinent information contained in the message into an appropriate temporary text file in a format that is readable by FLARE's database program. The temporary file in which to place the information is selected based on the type of message found in the mail file. The Parser is executed automatically by the Access database each time the database is started or when an engineer executes the importation routine from within the database to update the records. Appendix A contains the commented source code for the parser.

E. FLARE DATABASE

The database portion of FLARE's environment is implemented with Microsoft's Access database. The conceptual schema for this database is shown in the entity relationship model [Ref. 67] in Figure 12. The data requirements of the FLARE system are the storage of user needs and limitations, the storage of the required software system capabilities needed by users to solve their problems, and the storage of implementation domain information. The required implementation domain information consists of storage of engineer, programmer and design information. FLARE draws a distinction between programmers and engineers because they perform completely different functions and have different responsibilities.

This structure supports FLARE Teams by providing a means to manage the information gathered during requirements elicitation. Users have needs, and FLARE Team's are responsible for determining the requirements of software systems that will satisfy these needs. Programmers and other engineers will use the products produced by FLARE Teams and stored in the FLARE database to accomplish their respective tasks.

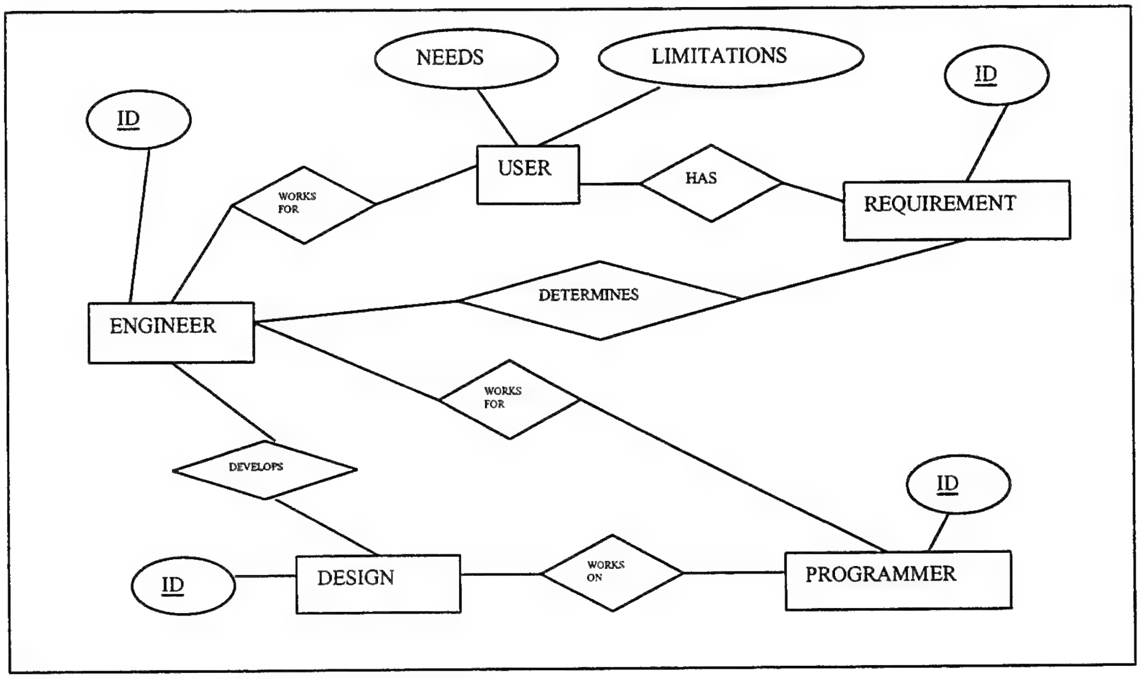


FIGURE 12. Entity Relationship Model.

1. Scheduling Algorithm

We developed and implemented a scheduling algorithm that automatically assigns tasks to programmers. The algorithm uses programmer's estimates (see Chapter IV-D) of the difficulty of translating a design specification module into a programming language implementation. The algorithm uses a greedy strategy [Ref. 68:p. 329]. It picks the lowest estimated time to implement a module and assigns that module to the programmer who made the estimate. The algorithm fails to produce optimal results in all cases, yet provides a close approximation. We find this acceptable knowing that the heuristic in which the algorithm determines a schedule is based on imprecise estimates. The queries that are listed in subsection D-3 constitutes the pseudo-code for this.

2. Database Tables

The database has the following tables. Information is entered into the tables automatically using the information created by the parser describe above.

a. *Table: assigned_modules*

This table contains the results of the scheduling algorithm that is implemented using the six queries described in section D-3.

b. *Table: bugReport*

This table is a collection of all bug reports.

c. *Table: changeRequest*

This table is a collection of all change requests.

d. *Table: engineers*

This table is a collection of all engineers working on a project. The primary key of this table also serves as the individual identification number for each engineer.

e. *Table: estimates*

This table is built using a query and has unique estimates. This makes it different from the imported_estimates table that likely contains duplications. For example, a programmer may mistakenly submit an estimate several times. This table lists the estimate only once, whereas the imported_estimates table lists each estimate submitted.

f. *Tables: Questions*

The evolution_questions, implementation_questions, maintenance_questions, design_questions, and req_questions tables store the questions submitted by various parties using the FLARE user interface. The tables also contain the answers to the questions.

g. *Table: imported_estimates*

This table contains all the estimates submitted by programmers. This table is used by the scheduling algorithm to build the assigned_modules table.

h. *Table: imported_requirements*

This table contains all the requirements submitted by engineers.

i. ***Table: information_requests***

This table stores all information requests submitted by users of the FLARE system.

j. ***Table: MNS***

This table contains the mission needs statement (MNS) that is the basis for the project being managed by the FLARE Team.

k. ***Table: modules***

This table contains the modules that have been approved by the project's manager. Each record in the table is a refined specification entered by some engineer using the FLARE user interface. Each entry in this table is an approved module. The scheduling algorithm uses this table to develop the assigned_modules table.

l. ***Table: programmers***

This table is a collection of all programmers working on a project. The primary key of this table also serves as the individual identification number for each programmer. Programmers enter this identification number to provide a means to verify input submitted using the HTML Forms described above.

m. ***Table: specification***

This table contains the raw specifications entered by engineers using the FLARE user interface. Managers and designers refine the specifications contained in this table and place the refined specifications in the modules table.

3. Database Queries

The database contains six queries that implement the scheduling algorithm described above.

a. ***Query: Remove duplicate estimates query***

This query eliminates duplications from the imported_estimates table.

b. ***Query: update matched estimates***

This query changes the assigned and committed fields of all records in the estimates Query table to yes. We do this to prevent programmers and modules from being assigned multiple times.

c. *Query: estimate Query*

This query builds a temporary list of unassigned programmers and modules that the assigned_Q query uses to match modules to programmers using the greedy strategy.

d. *Query: assigned_Q*

This query builds the assigned_modules table, which is the output of the scheduling algorithm.

e. *Query: modules Without matching assigned_modules*

This query identifies the modules that we marked as being assigned, but were not scheduled by the assigned_Q query. This happens because we only allow a programmer to work on one module at a time.

f. *Query: update unmatched modules*

This ensures all unassigned modules are marked as such.

4. Database Forms

We use the "import_data_form" to solve timing problems caused by the underlying operating system. For unexplained reasons, there is a noticeable delay in closing the temporary files created with the FLARE parser. This problem forced us to delay the importation of data into the database. We chose the timing mechanisms associated with Access' Forms to accomplish this. The form also acts as the driver that initiates the macros that automate the importation of information created by the FLARE parser. The remaining forms were designed to provide users of the database a more pleasing interface than that provided by the tables.

F. FLARE COMPARED TO OTHER DISTRIBUTED ENVIRONMENTS

We compare FLARE to the Dynamic Object-Oriented Requirements System (DOORS) and the Web Integrated Software Environment (WISE). DOORS is a mature requirements engineering tool designed specifically for requirements engineering. It provides a good base to assess the effectiveness of FLARE. WISE is a developmental tool that exploits Internet technologies to assist in the management of systems development. It provides a good base to assess the distributed aspects of FLARE.

1. DOORS

This is a full featured and comprehensive requirements engineering CASE tool capable of running in both the UNIX and personal computer environments [Ref. 69]. FLARE is a much simpler system and currently lacks the requirements tracing features of DOORS, as well as other features you would expect from a commercial tool. However, because FLARE uses Microsoft's Access 97 to implement its database functions, it has the full-featured power of a relational database and seamless compatibility with Microsoft's Office programs. Using Access, FLARE potentially can be developed to match the functionality of DOORS. Both DOORS and FLARE have the capability of importing requirements information from any text source.

2. WISE

This is an Internet based project management tool under development at West Virginia University [Ref. 70]. FLARE and WISE are similar in that they both use HTML Forms and a database to store submitted information. They both publish information for users to view. There are two major differences between the two tools. First, WISE runs on an Internet server and uses an online database. This gives it the ability to provide near real time data updates. FLARE can only update data with human intervention. Secondly, WISE offers online publication of project metrics. FLARE lacks this capability. The advantage FLARE has over WISE is the ease in which the environment can be established. Anyone with an electronic mail account and Access 97 can use FLARE, provided FormMail is installed and configured properly on their Internet Server.

G. SUMMARY

This Chapter introduced the new CASE tool FLARE. FLARE is a software-engineering environment created by using several distinct programs tied together with electronic mail. We described the Internet based user interface, the parsing program, and FLARE'S database that is implemented with Microsoft's Access relational database. We demonstrated how information is entered and retrieved from the system. Finally, we compared FLARE to WISE and DOORS, which are distributed software engineering CASE tools possessing far greater functionality than FLARE.

VI. CASE STUDY

A. INTRODUCTION

This case study is designed to show how DoD Software Engineers working in conjunction with contracted Software Engineers would use the FLARE CASE tool to aid in the development of a distributed requirements engineering environment. The other tools used in this case study are the modeling tool Rational Rose [Ref. 71] and the programming tool ObjectAda [Ref. 72].

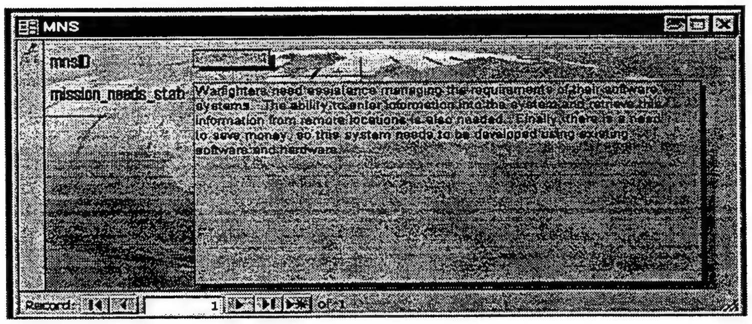
The methods, techniques and tools presented in this case study are applicable to purely in-house development projects, purely contracted development projects or a combination of the two.

B. MISSION NEEDS STATEMENT

The event that initiates the software development process is a mission needs statement developed by some commander in the DoD. The mission needs statement that triggered the development of our requirements engineering environment is:

> Warfighters need assistance managing the requirements of their software systems. The ability to enter information into the system and retrieve this information from remote locations is also needed. Finally, there is a need to save money, so this system needs to be developed using existing software and hardware.

The command's software engineering team assisted in the development of the mission needs statement. Once the commander approved the MNS statement, the FLARE Team entered the MNS into the FLARE environment (Figure 13). One advantage of using Microsoft's Access is the ability to use the



mnsID	1
mission_needs_stat	Warfighters need assistance managing the requirements of their software systems. The ability to enter information into the system and retrieve this information from remote locations is also needed. Finally, there is a need to save money, so this system needs to be developed using existing software and hardware.

Figure 13. Mission Needs Statement Form.

clipboard feature to cut from one application and paste it into another. We used this technique to enter the mission needs statement into the MNS Form found in FLARE's database.

C. REQUIREMENTS IDENTIFICATION

The FLARE Team performed requirements elicitation and identification activities. Team members' requirements perception was enhanced by their formal software engineering education and their problem domain experience. The following are the initial requirements identified by the team.

1. Initial Requirements

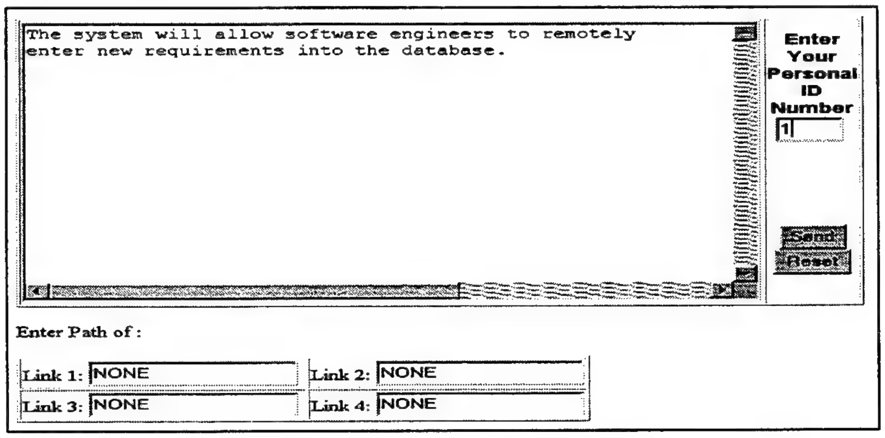
a. The system will allow Software Engineers to remotely enter new requirements into the database.

b. The system will allow Software Engineers to remotely view all approved requirements.

c. The system will allow multiple Software Engineers to simultaneously enter and retrieve information.

2. Requirements Entered Using a Form

The Software Engineers using the Internet browser user interface as shown in Figure 14 enter the requirements into the FLARE system. The FLARE system requires that engineers enter only one


The system will allow software engineers to remotely enter new requirements into the database.

Enter Your Personal ID Number: 1

Enter Path of :

Link 1: NONE
Link 2: NONE
Link 3: NONE
Link 4: NONE

FIGURE 14. Requirement Entered Into the Flare System.

requirement in a form at a time. This ensures that each requirement is given a unique identification number used to track the requirement throughout the software development process.

D. SYSTEM DESIGN

The FLARE Team made an object model (Figure 15) of a prototype system that would satisfy the

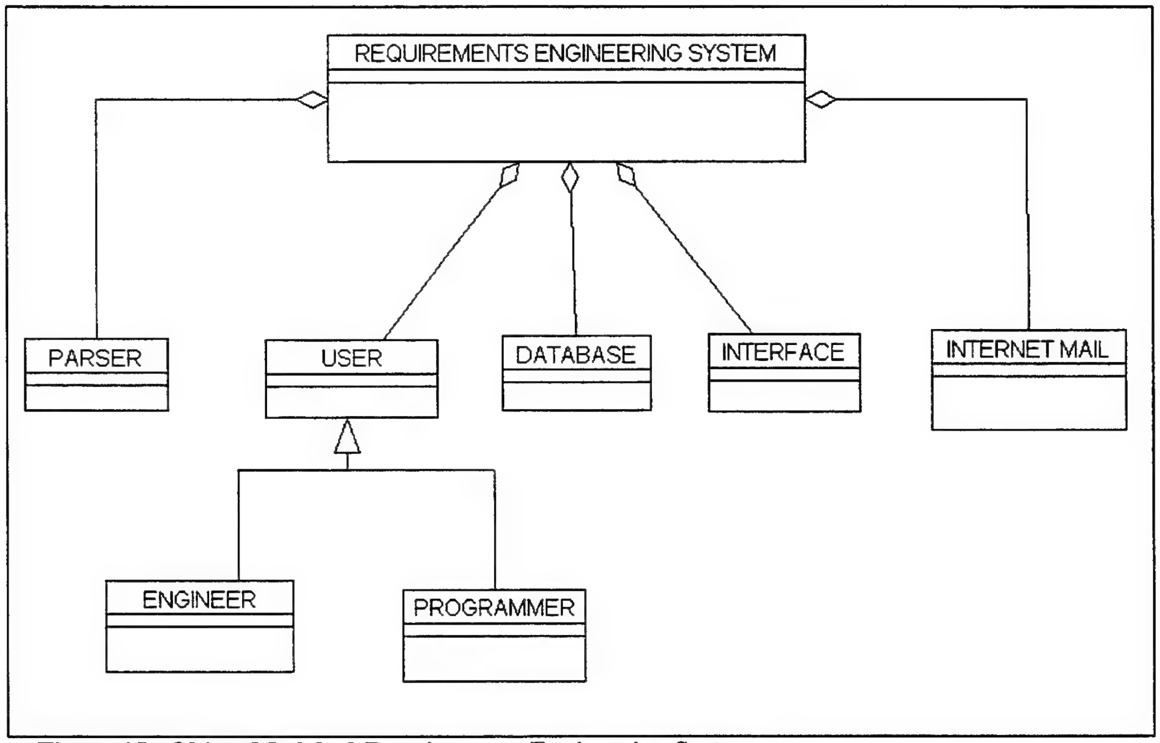
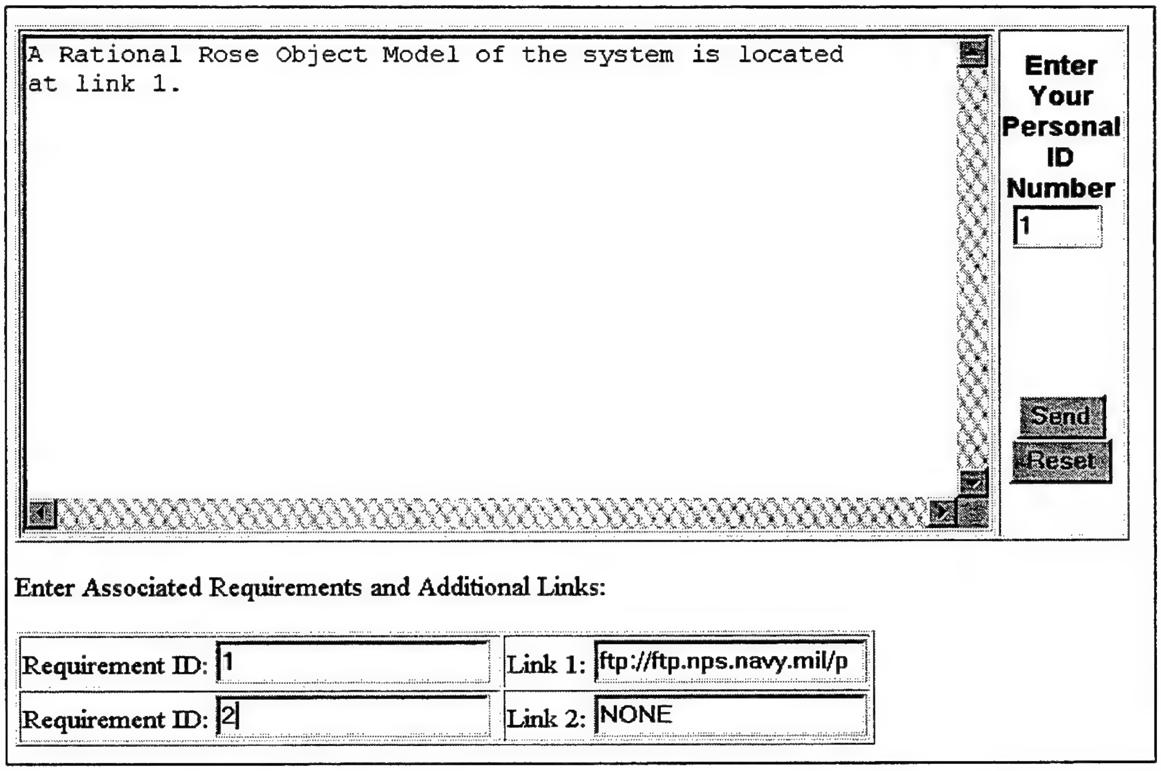


Figure 15. Object Model of Requirements Engineering System.

requirements elicited from the problem domain using Rational Rose. Rational Rose produces informal models that are in graphical form. Graphic files can not be pasted into the text box found in the FLARE's user interface. Each of the FLARE input forms has fields that allow engineers to enter hyperlinks to objects such as graphical object models.

Figure 16 shows how the design team used FLARE's specification form to enter the hyperlink to the graphical object model into the FLARE system. Notice that the link is to an ftp site. This will allow anyone in the team to download the graphic file and view it using his or her copy of Rational Rose. This is an example of how FLARE extends traditional software engineering environments. The same technique


```
A Rational Rose Object Model of the system is located
at link 1.
```

Enter Your Personal ID Number: 1

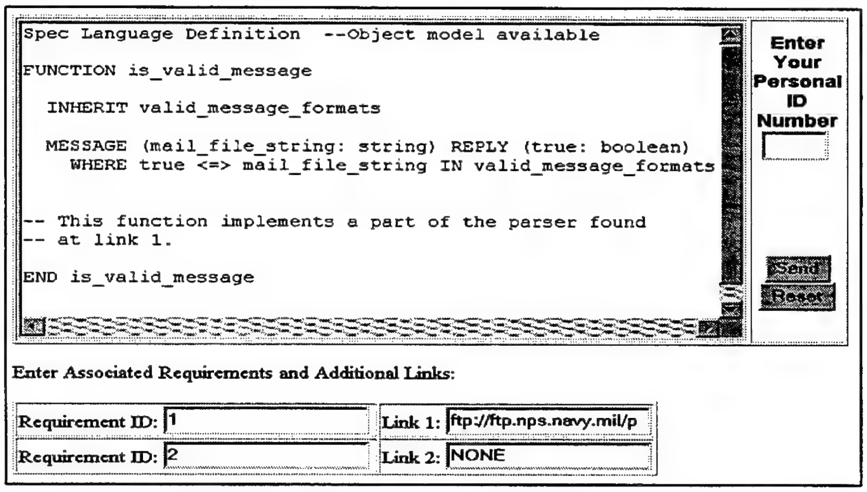
Enter Associated Requirements and Additional Links:

Requirement ID: 1	Link 1: ftp://ftp.nps.navy.mil/p
Requirement ID: 2	Link 2: NONE

Figure 16. A Specification Entry Form Used to Enter A Graphical Object Model.

the engineering team used to extend the environment of Rational Rose can be used to extend any engineering environment.

Figure 17 shows another example of how the team used the specification entry form to enter a specification into the system. This time the team defined a Spec Language definition of a function that determines if a message contained in an electronic mail file is a valid. If the message is valid a true value is returned. This function is the specification of an operation contained in the "PARSER" object found in figure 15. The designer chose to include the requirements that the specification helps to satisfy, as well as a hyperlink to the object model of the system that contains the "PARSER" object. Notice that the personal ID field is not filled in. If the user were to press the send button with this field left blank, they would receive an error message generated by FormMail's CGI script indicating that a required field was left blank.



```
Spec Language Definition  --Object model available

FUNCTION is_valid_message

  INHERIT valid_message_formats

  MESSAGE (mail_file_string: string) REPLY (true: boolean)
    WHERE true <=> mail_file_string IN valid_message_formats

-- This function implements a part of the parser found
-- at link 1.

END is_valid_message
```

Enter Associated Requirements and Additional Links:

Requirement ID:	1	Link 1:	ftp://ftp.nps.navy.mil/p
Requirement ID:	2	Link 2:	NONE

Figure 17. Specification Entry Form Used To Enter a Text Specification.

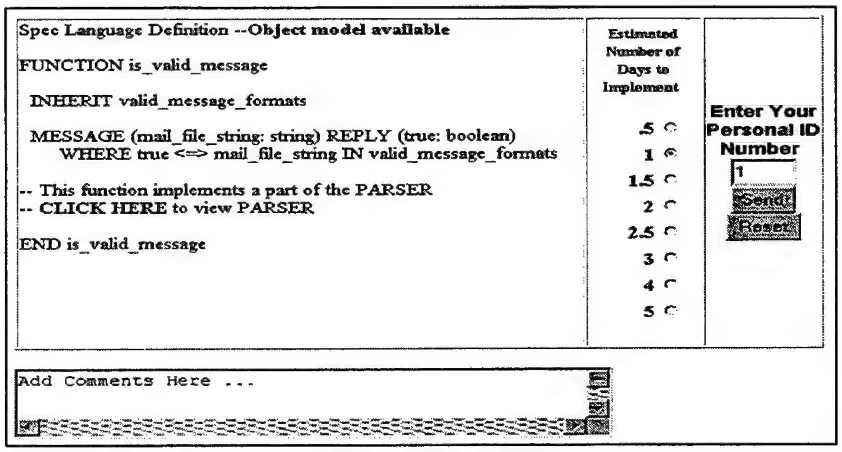
Programmers can begin to estimate how long they think it will take them to implement a module as soon as the design team enters the first specification into the FLARE system.

E. IMPLEMENTATION

Figure 18 shows the FLARE form used by programmers to enter their implementation time estimates. They use the FLARE user interface to browse the set of unassigned modules and are required to estimate how long they think it would take them to implement each specification. Notice the bold text "CLICK HERE" in the form. This is a link to the object model file containing the "PARSER" object. This feature is provided to give the programmer additional information to use in order to make a better estimate of the time needed to implement the specification. The design team used their Internet browser's HTML editor to easily create the form shown in Figure 18. A designer accomplished this task in less than five minutes.

Once idle programmers have completed their estimates, a manager uses FLARE to determine an implementation schedule. FLARE generates a list showing the assignment of modules to programmers. The database generates an HTML file depicting the assignments, which management can post to allow programmers see what module they have been tasked to implement. The programmers in this case study use the Ada 95 programming language to implement design specifications. They use the programming environment created by OjectAda, and they augment this environment with FLARE.

By using FLARE, the programmers have access to all the requirements and specifications associated with the module they are implementing. This allows them to find information about the module that they have been tasked to implement if they encounter any ambiguities in the specification. The



```
Spec Language Definition --Object model available

FUNCTION is_valid_message

 INHERIT valid_message_formats

 MESSAGE (mail_file_string: string) REPLY (true: boolean)
     WHERE true <=> mail_file_string IN valid_message_formats

-- This function implements a part of the PARSER
-- CLICK HERE to view PARSER

END is_valid_message
```

Estimated Number of Days to Implement

.5 [✓ 1] 1.5 2 2.5 3 4 5

Enter Your Personal ID Number: 1

Figure 18. Programmer Estimate Entry Form.

FLARE environment also gives programmers the option of asking implementation questions. The questions are imported into the database where management can review them in an effort to determine if patterns exist that may indicate a need to improve one or more design processes.

F. MAINTENANCE

The maintenance phase begins after the system is delivered to the user. FLARE offers a means to easily input bug reports into the system. This is accomplished using the Internet user interface. The software engineering team and users use the Bug Report Form to input a description of any problems found into the FLARE system. This allows engineers to easily access the complete record of bug reports. The team uses the set of bug reports to help them find indicators of design errors, coding errors, or a combination of the two.

G. NEW REQUIREMENTS ARE IDENTIFIED; THE SYSTEM EVOLVES

FLARE provides a way for users to input proposed changes into the system. This is a needed feature because users will likely want to improve or add additional functionality to the software system.

A user accomplishes this by selecting the "Enter Change Request" option found under the "MAINTENANCE" menu located on FLARE's user interface. All change requests are imported into the database. FLARE Teams use this information in their requirements engineering efforts. The organization's commander would approve or disapprove any recommended changes because change requests could alter or extend the original mission needs statement. Any approved changes would start the development process over again: additional requirements would be identified, the design would be altered and changes would be implemented.

H. SUMMARY

This case study showed how a team of Software Engineers could use FLARE to enhance traditional software engineering environments, as well as assist in the management and communication of requirements. The study started by showing how a mission needs statement is entered into the FLARE system. We showed how FLARE easily extends the reach of Software Engineers by transforming them from engineers working on isolated systems to engineers working in a distributed engineering environment.

FLARE is not a silver bullet that will cure the software engineering community's requirements engineering problems. However, the case study shows that FLARE can enhance traditional software engineering tools and environments. Its distributed features allow Team members to input and access requirements information in real-time anywhere access to the Internet can be found. FLARE's use of Internet technologies also allows all stakeholders and developers to participate in requirements engineering activities. This likely will increase the quality of the requirements engineering process. This increased quality is the factor that will likely cause a reduction in the number of changes in requirements caused by CINC turnover. New CINCs quickly formulate opinions of the quality of personnel and systems within the command. By involving stakeholders in the requirements engineering process, they are more likely to have a positive view of the software system. Hence, they will communicate this positive view when they speak to new CINCs.

VII. CONCLUSIONS AND FUTURE WORK

A. RESEARCH CONTRIBUTIONS

We developed a feasible course of action that DoD decision-makers can use while formulating ways to reduce the number of unwanted, unneeded and unusable software systems. This course of action is based on the formation of special staffs by each of the geographic and functional CINCs. These staffs would be composed of military officers possessing advanced software engineering degrees. Their mission would be to conduct requirements engineering for their command. We explained how these teams would naturally and easily fit into the current acquisition process.

We developed the CASE tool FLARE. FLARE is a requirements engineering environment composed of commercial off the shelf (COTS) and government of the shelf (GOTS) software tools tied together by electronic mail and a parsing program that we developed.

B. SUGGESTIONS FOR FUTURE RESEARCH

1. Proof of Concept Experiment for the Special Staff

We suggest the identification of a CINC who is willing to implement the course of action detailed in Chapter three of this thesis and that the results produced by the team be compared to results produced by the methods used by the remaining CINCs. This research would accomplish two things. First, it would assess whether or not the course of action provided in this thesis is worth pursuing. Secondly, the current state of the software engineering capabilities of our major commands would be clarified.

2. Requirements Tracing Features

FLARE's requirements tracing features are primitive. Even though each requirement receives a unique identification number, FLARE does not automatically track this requirement throughout the development process, rather it relies on engineers "tagging" each new product with the appropriate requirement identification number. Automation of this tagging process would eliminate possibilities of entering incorrect tags, and could potentially allow engineers to look at any object produced in the development process and extract the associated requirement information.

3. Report Generation Enhancement

Microsoft's Access database provides the ability to automatically generate HTML files based on the information submitted by the software engineering team. These files are crude. Research to develop a more sophisticated HTML file generator is needed.

4. Module Assignment Algorithm Enhancement

The greedy strategy used by the module assignment algorithm does not guarantee an optimal solution. The development of an algorithm that would always produce an optimal solution is needed.

APPENDIX A. ELECTRONIC MAIL FILE PARSER SOURCE CODE

```
--------------------------------------------------------
-- Name:  Main_FLARE_parser.adb
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program takes a Netscape mail file (Inbox) and
--   parses it to extract information provided by programmers
--   using a cgi script.  It writes the required information to an
--   output file, which is incorporated into a database.
--------------------------------------------------------

with Text_IO, Ada.Integer_Text_IO, variables, process_est_pkg, process_req_pkg;
use  Text_IO, Ada.Integer_Text_IO, variables, process_est_pkg,  process_req_pkg;

procedure FLARE is

-- declare file objects to use in the processing of the input file
 indata: File_Type;
 req_outdata: File_Type;                          --file new requirements are added to
 est_outdata: File_Type;                          --file new estimates are added to
 spe_outdata: File_Type;                          --file new requirements are added to
 bug_outdata: File_Type;                          --file new estimates are added to
 cha_outdata: File_Type;                          --file new requirements are added to
 f_outdata: File_Type;                            --file new requests are added to
 requirements_questions_outdata: File_Type;       --file new estimates are added to
 design_questions_outdata: File_Type;             --file new requirements are added to
 implementation_questions_outdata: File_Type;     --file new estimates are added to
 maintenance_questions_outdata: File_Type;        --file new requirements are added to
 evolution_questions_outdata: File_Type;          --file new requests are added to

begin

-- open the input file
 open (File => indata, mode => In_File, Name => "C:\flare\inbox");

-- create the output files
 create (File => req_outdata, mode => Out_File, Name => "c:\flare\temp_req.txt");
 create (File => est_outdata, mode => Out_File, Name => "c:\flare\temp_est.txt");
 create (File => spe_outdata, mode => Out_File, Name => "c:\flare\temp_spe.txt");
 create (File => bug_outdata, mode => Out_File, Name => "c:\flare\temp_bug.txt");
 create (File => cha_outdata, mode => Out_File, Name => "c:\flare\temp_cha.txt");
 create (File => f_outdata, mode => Out_File, Name => "c:\flare\temp_f.txt");
 create (File => requirements_questions_outdata, mode => Out_File, Name =>
          "c:\flare\temp_req_questions.txt");
 create (File => design_questions_outdata, mode => Out_File, Name =>
          "c:\flare\temp_design_questions.txt");
 create (File => implementation_questions_outdata, mode => Out_File, Name =>
          "c:\flare\temp_implementation_questions.txt");
 create (File => maintenance_questions_outdata, mode => Out_File, Name =>
          "c:\flare\temp_maintenance_questions.txt");
```

```
-------------------------------------------------------
-- Name:  Main_FLARE_parser.adb (continued)
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program takes a netscape mail file (infile) and
--   parses it to extract information provided by programmers
--   using a cgi script.  It writes the required information to an
--   output file, which is incorporated into a database.
-------------------------------------------------------

create (File => evolution_questions_outdata, mode => Out_File, Name =>
          "c:\flare\temp_evolution_questions.txt");

-- process the input file
while not End_of_File(File => indata) loop

 --assume the next string is of interest
 is_desired:= true;
 get_line(file => indata, item => string_buffer, last => last_char);

 --this strips a character off to expose the End of File symbol
 get(file => indata, Item => nextchar);

 --test to see if the line is larger than our deliminator
 if Last_Char > size1 then

  --check to see if the line contains our deliminator
  for I in 1..size1 loop
   if string_buffer(I) /= flag(I) then
    is_desired := false;      -- the line was not of interest
    exit;
   end if;
  end loop;

  --if line was of interest then add input to the output file
  if is_desired then

   if string_buffer(size8) = ')' then
     process_est (est_outdata, indata);
   elsif string_buffer(size8) = 'R' then  --process a requirement
     process_req (req_outdata, indata);
   elsif string_buffer(size8) = 'S' then  --process a specification
    process_req (spe_outdata, indata);
   elsif string_buffer(size8) = 'B' then  --process a bug report
    process_req (bug_outdata, indata);
   elsif string_buffer(size8) = 'C' then  --process a change request
    process_req (cha_outdata, indata);
   elsif string_buffer(size8) = 'F' then  --process a general information request
    process_req (f_outdata, indata);

   -- process questions
   elsif string_buffer(size8) = 'r' then  --process a requirement
     process_req (requirements_questions_outdata, indata);
```

```
----------------------------------------------------------
-- Name:  Main_FLARE_parser.adb (continued)
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program takes a netscape mail file (infile) and
--   parses it to extract information provided by programmers
--   using a cgi script.  It writes the required information to an
--   output file, which is incorporated into a database.
----------------------------------------------------------


   elsif string_buffer(size8) = 'd' then  --process a specification
      process_req (design_questions_outdata, indata);
    elsif string_buffer(size8) = 'i' then  --process a bub report
      process_req (implementation_questions_outdata, indata);
    elsif string_buffer(size8) = 'm' then  --process a change request
      process_req (maintenance_questions_outdata, indata);
    elsif string_buffer(size8) = 'e' then  --process a general information request
      process_req (evolution_questions_outdata, indata);
    end if;
  end if; -- end adding information to output file


 end if; -- end checking the line

end loop; -- all lines have been processed

exception

 --this exception is allways raised because we removed the EOF special character
 --with the get(file => indata, Item => nextchar); command above.
 when End_Error =>
  Put("You made it to the end of file : ");

close(File => indata);
close(File => est_outdata);
close(File => req_outdata);
close(File => spe_outdata);
close(File => bug_outdata);
close(File => cha_outdata);
close(File => f_outdata);
close(File => requirements_questions_outdata);
close(File => design_questions_outdata);
close(File => implementation_questions_outdata);
close(File => maintenance_questions_outdata);
close(File => evolution_questions_outdata);

--create (File =>  cha_outdata, mode => Out_File, Name => "C:\program
files\netscape\users\Leonard\mail\inbox");
--close(File => cha_outdata);

end FLARE;
```

```
-- Name:  get_clip_pkg.ads
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


with Text_IO, Ada.Integer_Text_IO, variables;
use  Text_IO, Ada.Integer_Text_IO, variables;


package get_clip_pkg is

procedure get_clip (req_outdata, indata : File_Type);

end get_clip_pkg;



-- Name:  get_clip_pkg.adb
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


package body get_clip_pkg is

procedure get_clip  (req_outdata: in File_Type; indata : in File_Type) is

--declare a variable to use to strip delimiter from input string.
clip_holder : string (1..8);

begin

   --strip delimiter.
   get(file => indata, item => clip_holder);
   --read the hyperlink
   get_line(file => indata, item => string_buffer, last => last_char);

   --place delimiter in the output file.
   put(file => req_outdata, item => "^");
   put("  ");
   Put(item => string_buffer(1..last_char));
   put (File => req_outdata, Item => string_buffer(1..last_char));

 end get_clip;

 end get_clip_pkg;
```

```
-- Name:  process_dat_pkg.ads
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


with Text_IO, Ada.Integer_Text_IO, variables;
use  Text_IO, Ada.Integer_Text_IO, variables;


package process_date_pkg is

procedure process_date (outdata, indata : File_Type; begin_date : integer);

end process_date_pkg;



-- Name:  process_date_pkg.adb
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


package body process_date_pkg is

procedure process_date (outdata : in File_Type; indata  : in  File_Type;  begin_date : in integer) is

begin

 --process the delimiter of the date first.
 field1 := 1;

 --extract date from input file and change format so it can
 --be read by database program.
 for I in begin_date..last_char - 1 loop

   case field1 is

     when 1 => month(index) := string_buffer(I);    --strip delimiter
       index := index + 1;
       if string_buffer(I) = space(1) then
         field1 := field1 + 1;
         index := 1;
         month := "         ";
       end if;
```

```
-- Name:  process_date_pkg.adb (continued)
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


    when 2 =>  month(index) := string_buffer(I);  --get month
     index := index + 1;
     if string_buffer(I) = space(1) then
       field1 := field1 + 1;
       index := 1;

       case month(1) is

         when 'J' | 'j' =>
           if month(2) = 'A' then
             month_number := 1;
           elsif month (2) = 'a' then
             month_number := 1;
           elsif month(3) = 'N' then
             month_number := 6;
           elsif month(3) = 'n' then
             month_number := 6;
           else month_number := 7;
           end if;

         when 'F' | 'f' => month_number := 2;

         when 'M' | 'm' =>
           if month(3) = 'R' then
             month_number :=3;
           elsif month(3) = 'r' then
             month_number := 3;
           else month_number := 5;
           end if;

         when 'A' | 'a' =>
           if month(2) = 'P' then
             month_number := 4;
           elsif month(2) = 'p' then
             month_number := 4;
           else month_number := 8;
           end if;

         when 'S' | 's' => month_number := 9;
         when 'O' | 'o' => month_number := 10;
         when 'N' | 'n' => month_number := 11;
         when 'D' | 'd' => month_number := 12;
         when others => null;
       end case;
```

```
-- Name:  process_date_pkg.adb (continued)
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


      put(file => outdata, item => "^");
      put(month_number);
      put(file => outdata, item => month_number);
      put("/");
      put(file => outdata, item => "/");
      month := "         ";
     end if;

    when 3 => day(index) := string_buffer(I);  --get day
     index := index + 1;
      if string_buffer(I) = space(1) then
      day(index - 2) := space(1);
      field1 := field1 + 1;
      index := 1;
      if day(2) /= ' ' then
       put(day(day'first..2));
       put(file => outdata, item => day(day'first..2));
      else
       put(day(1));
       put(file => outdata, item => day(1));
      end if;
      put("/");
      put(file => outdata, item => "/");
     end if;

    when 4 => year(index) := string_buffer(I);  --get year
     index := index + 1;
     if string_buffer(I) = space(1) then
      field1 := field1 + 1;
      index := 1;
      put(year);
      put(file => outdata, item => year);
      put(" ");
      --pad the date if needed
      if day(2) = ' ' then
       put(file => outdata, item => "  ");
      else
       put(file => outdata, item => " ");
      end if;
      day := "  ";
      year := "    ";
     end if;

    when others => null;
```

```
-- Name:  process_date_pkg.adb (continued)
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


    end case;

  end loop;

end process_date;

end process_date_pkg;



-- Name:  process_est_pkg.ads
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts information
-- from it, and writes the desired information to an output file.


with Text_IO, Ada.Integer_Text_IO, variables, process_date_pkg;
use  Text_IO, Ada.Integer_Text_IO, variables, process_date_pkg;


package process_est_pkg is

procedure process_est (est_outdata, indata : File_Type);

end process_est_pkg;
```

```
-- Name:  process_est_pkg.adb
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts programmer
-- estimate information
-- from it, and writes the desired information to an output file.


package body process_est_pkg is

procedure process_est (est_outdata : in File_Type; indata : in  File_Type) is

begin

   --process the date
   process_date(est_outdata, indata, start_date);

   skip_line(file => indata, Spacing => 2);

   --get the programmer identification from the input file
   get(file => indata, item => strip_string1);
   get_line(file => indata, item => programmer_id, last => last_char);

   --write ID into the output file
   put("  ");
   put (File => est_outdata, Item => "  ");
   Put(item => programmer_id(1..last_char));
   put (File => est_outdata, Item => programmer_id(1..last_char));

   if fill1 = last_char then Put( "     ");
      Put(File => est_outdata, item => "     ");
     elsif fill2 = last_char then Put( "    ");
      Put(File => est_outdata, item => "    ");
     elsif fill3 = last_char then Put( "  ");
      Put(File => est_outdata, item => "  ");
     elsif fill4 = last_char then Put( " ");
      Put(File => est_outdata, item => " ");
   end if;

   --extract the estimated days to implement from the input file.
   get(file => indata, item => strip_string2);
   get_line(file => indata, item => days_to_implement, last =>
     last_char);
   Put( "  ");
   Put(File => est_outdata, item => "    ");
   put(item => days_to_implement(1..last_char));
   Put(File => est_outdata, item => days_to_implement(1..last_char));
```

```
-- Name:  process_est_pkg.adb (continued)
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts programmer
-- estimate information
-- from it, and writes the desired information to an output file.


 if fill1 = last_char then Put( "  ");
       Put(File => est_outdata, item => "  ");
     elsif fill2 = last_char then Put( " ");
       Put(File => est_outdata, item => " ");
     elsif fill3 = last_char then Put( "");
       Put(File => est_outdata, item => "");
   end if;

   --get the module identification from the input file.
   get(file => indata, item => strip_string3);
   get_line(file => indata, item => specification_module_name, last =>
     last_char);

   put(item => specification_module_name(1..last_char));
   Put(File => est_outdata, item =>
     specification_module_name(1..last_char));

   new_line;
   new_line(file => est_outdata, Spacing=> 1);

end process_est;

end process_est_pkg;



-- Name:  process_req_pkg.ads
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts a requirement
-- from it, and writes the requirement to an output file.


with Text_IO, Ada.Integer_Text_IO, variables, process_date_pkg;
use  Text_IO, Ada.Integer_Text_IO, variables, process_date_pkg;
with get_clip_pkg; use get_clip_pkg;

package process_req_pkg is

procedure process_req (req_outdata, indata : File_Type);

end process_req_pkg;
```

```
-- Name:  process_req_pkg.adb
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts a requirement
-- from it, and writes the requirement to an output file.


package body process_req_pkg is

procedure process_req (req_outdata : in File_Type; indata : in  File_Type) is

size_of_delimiter : integer := 7;
number_of_clips : integer := 4;
delimiter : string (1..size_of_delimiter) := "zend: Z";
comments_holder : string (1..10);
end_comments_holder : string (1..20);
not_end : boolean := true;   --used to determine end of requirement

begin

   --extract the date first.
   process_date(req_outdata, indata, (start_date + 1));

   skip_line(file => indata, Spacing => 2);

   --get the number of video clips available.
   for I in 1..number_of_clips loop
      get_clip(req_outdata, indata);
   end loop;

   --get the programmer identification number.
   get(file => indata, item => strip_string1);
   get_line(file => indata, item => programmer_id, last => last_char);
   put(file => req_outdata, item => "^");
   put("  ");
   Put(item => programmer_id(1..last_char));
   put (File => req_outdata, Item => programmer_id(1..last_char));

   --place a delimiter in the output file
   put(file => req_outdata, item => "^");

   --mark the beginning of the requirement being extracted.
   put(file => req_outdata, item => """");

   get_line(file => indata, item => strip_string4, last => last_char);
   --process lines until we reach the end of the requirement.
   while not_end loop

     get_line(file => indata, item => string_buffer, last =>
       last_char);
```

```
-- Name:  process_req_pkg.adb (continued)
-- Date:  29 March, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This program reads an input file, extracts a requirement
-- from it, and writes the requirement to an output file.


    -- test to see if we are at the end of the requirements
    if last_char = size_of_delimiter then

      for I in 1..size_of_delimiter loop
        if string_buffer(I) /= delimiter(I) then
          null;
        else
          not_end := false;
        end if;
      end loop;

    end if;

    -- put the line in output file if not at end
    if not_end then

      -- remove all " from the text
      -- this is required because of how the database determines
      -- what a character sting is.  Replace each occurrence with a
      -- blank space.
      for I in 1..last_char loop
        if string_buffer(I) = '"' then
          string_buffer(I) := ' ';
        end if;
      end loop;

      --write the cleaned line to the output file.
      Put(item => string_buffer(1..last_char));
      put (File => req_outdata, Item => string_buffer(1..last_char));
      new_line(File => req_outdata);

    end if;

  end loop;

  --this is needed to show the end of the string in the database.
  put(file => req_outdata, item => """");
  new_line(File => req_outdata);

end process_req;

end process_req_pkg;
```

```
-- Name: variables.ads
-- Date:  4 April, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This file holds global variables to use with a
-- netscape mail parser program.


package variables is

-- declare variable used for formating
  month_number : integer;
  count : integer := 1;
  start_date : integer := 61;
  index : integer := 1;
  field1 : integer := 1;
  fill0 : natural := 0;
  fill1 : natural := 1;
  fill2 : natural := 2;
  fill3 : natural := 3;
  fill4 : natural := 4;
  fill5 : natural := 5;

-- declare size of a buffer to hold a line from the input file
  size_of_buffer : integer := 100;

-- declare variables to hold fields taken from the input file
  size1 : integer := 55;
  size2 : integer := 4;
  size3 : integer := 3;
  size4 : integer := 5;
  size5 : integer := 31;
  size6 : integer := 39;
  size7 : integer := 20;
  size8 : integer := 56;  --this is where the delimiter is
  size9 : integer := 10;

-- declare strings used to strip unwanted information from an input -
-- line
  strip_string1 : string (1..size5);
  strip_string2 : string (1..size6);
  strip_string3 : string (1..size7);
  strip_string4 : string (1..size9);

-- declare string to hold an input line
  string_buffer : String (1..size_of_buffer);

-- declare strings to hold field information from the input file
  space : string(1..1) := " ";
  specification_module_name : String (1..size4);
  days_to_implement : String (1..size3);
  programmer_id : String (1..size4);
```

```
-- Name: variables.ads (continued)
-- Date:  4 April, 1997
-- Author:  Anthony E. Leonard
-- Purpose:  This file holds global variables to use with a
-- netscape mail parser program.


  month : string (1..10) := "          ";
  day : string (1..4) := "    ";
  year : string (1..5) := "     ";

-- declare strings to hold the names of input and out files
  infile_name : string(1..size_of_buffer);
  output_file_name : string(1.. size_of_buffer);

-- declare this string, which is a special string used to find the
-- the text that we are interested in that is contained in the input
-- file
  flag : string (1..size1) :=
    "elow is the contents of a form.  It was submitted by  (";

-- declare a variable to hold a character taken off to expose the EOF
-- signature
  nextchar : character;

-- declare a variable to hold the length of a string
  Last_Char : Natural;
  Last_Char2 : Natural;

-- declare a variable to show if two strings were equal
  is_desired : Boolean := true;

end variables;
```

APPENDIX B. FLARE INTERFACE SOURCE CODE

```
<!------------------------------------------------------------->
<!-- Name:  requirement_entryform.html  >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard  >
<!-- Purpose:  This file is used by users to input requirements >
<!--   into the FLARE system.  The file uses a form to call the >
<!--   FormMail.pl program located in the cgi-bin directory on  >
<!--   the server.  The file was partially produced using the editor  >
<!--  found in Netscape Navigator version 3.01. >
<!------------------------------------------------------------->



<HTML>
<HEAD>
<TITLE>Spec form</TITLE>
<META NAME="GENERATOR" CONTENT="Mozilla/3.01Gold (Win95; U)
[Netscape]">
</HEAD>
<BODY>

<CENTER><P>
<!put the URL of FormMail here >
<FORM METHOD=POST ACTION="http://web.nps.navy.mil/cgi-bin/FormMail.pl">
</P></CENTER>

<TABLE BORDER=1 >
<CAPTION></CAPTION>

<TR>
<TD ALIGN=CENTER VALIGN=TOP COLSPAN="" NOWRAP WIDTH="">
<TEXTAREA NAME="comments" ROWS=15 COLS=60>Enter A New Requirement
Here...
</TEXTAREA></TD>

<TD align=center valign=middle>
<CENTER><P><B><FONT FACE="arial">Enter<BR>
Your<BR>
Personal<BR>
ID<BR>
Number</FONT></B><BR>

<!The input size limits the number of characters allowed in the
personal ID field>
<INPUT SIZE=5 MAXLENGTH=5 NAME="Enter Your Personal ID Number">
<BR>
<BR>
```

```
<!------------------------------------------------------->
<!-- Name:  requirement_entryform.html (continued)  >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard  >
<!-- Purpose:  This file is used by users to input requirements >
<!--   into the FLARE system.  The file uses a form to call the >
<!--   FormMail.pl program located in the cgi-bin directory on  >
<!--   the server.  The file was partially produced using the editor  >
<!--  found in Netscape Navigator version 3.01. >
<!------------------------------------------------------->



<!These two values are used by the parser to determine the end of
message>

<INPUT type="hidden" name="zend" value="Z"><BR>
<INPUT type="hidden" name="zzend" value="Zsa"><BR>
<BR>

<!This is where you enter the address to mail the input to>
<INPUT type="hidden" name="email" value="R"><BR>
<INPUT type="hidden" name="recipient" value="Leonard@cs.nps.navy.mil">

<!These fields are required.  If they are changed the parser must also
be changed>

<INPUT type=hidden name="required"
value="Enter Your Personal ID Number,Clip 1,Clip 2,Clip 3,Clip 4">
<INPUT type=hidden name="sort" value="alphabetic">
<INPUT TYPE=SUBMIT VALUE="Send">
<BR>
<INPUT TYPE=RESET></P></CENTER>
</TD>
</TR>
</TABLE>

<P>Enter Path of :</P>
<TABLE BORDER=1 >
<TR>
<TD>Link 1: <INPUT SIZE=20 NAME="Clip 1" value="NONE"></TD>
<TD>Link 2: <INPUT SIZE=20 NAME="Clip 2" value="NONE"></TD></TR>
<TR>
<TD>Link 3: <INPUT SIZE=20 NAME="Clip 3" value="NONE">&nbsp;</TD>
<TD>Link 4: <INPUT SIZE=20 NAME="Clip 4" value="NONE"></TD>
</TR>
</TABLE>


<!This is formatting information.  Also contains the URL FormMail will
goto>
<!once the send button is pressed. >
<P>
```

```
<!---------------------------------------------------------->
<!-- Name:  requirement_entryform.html (continued)  >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard  >
<!-- Purpose:  This file is used by users to input requirements >
<!--   into the FLARE system.  The file uses a form to call the >
<!--   FormMail.pl program located in the cgi-bin directory on  >
<!--   the server.  The file was partially produced using the editor  >
<!--  found in Netscape Navigator version 3.01. >
<!---------------------------------------------------------->



<BR>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
&nbsp;&nbsp;<INPUT type=hidden name="redirect"
value="http://web.nps.navy.mil/~aeleonar/FLARE/DEMOfrontpage.html"><BR>
</P>

<P></FORM></P>

</BODY>
</HTML>



<!---------------------------------------------------------->
<!-- Name:  DEMOfrontpage.html >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard >
<!-- Purpose:  This file creates the initial user interface. >
<!--   It has four pull-down menus: REQUIREMENTS, DESIGN, >
<!--   IMPLEMENTATION, AND MAINTENANCE. >
<!--   The file was partially produced using the editor >
<!--   found in Netscape Navigator version 3.01. >
<!---------------------------------------------------------->




<HTML>
<HEAD>
<TITLE>REQUIREMENTS</TITLE>
<SCRIPT LANGUAGE='JavaScript'>

<!These functions open the apporiated page selected by the user using
pulldown menus>

function switch_page1() {
      if (document.menuform.F1.selectedIndex == 0) location =
            'requirement_entryform.html';
```

<!-->
<!-- Name: DEMOfrontpage.html (continued) >
<!-- Date: 29 March, 1997 >
<!-- Author: Anthony E. Leonard >
<!-- Purpose: This file creates the initial user interface. >
<!--   It has four pull-down menus: REQUIREMENTS, DESIGN, >
<!--   IMPLEMENTATION, AND MAINTENANCE. >
<!--   The file was partially produced using the editor >
<!--   found in Netscape Navigator version 3.01. >
<!-->

```
        else if (document.menuform.F1.selectedIndex == 1) location =
                'requirement_entryform.html';

        else if (document.menuform.F1.selectedIndex == 2) location =
                'Requirements_1.html';

        else if (document.menuform.F1.selectedIndex == 3) location =
                'requirement_question.html';

        else if (document.menuform.F1.selectedIndex == 4) location =
                'requirements_questions_1.html';

        else if (document.menuform.F1.selectedIndex == 5) location =
                'MNS_1.html';

}

function switch_page2() {

        if (document.menuform.F2.selectedIndex == 0) location =
                'specification_entryform.html';

        else if (document.menuform.F2.selectedIndex == 1) location =
                'specification_entryform.html';

        else if (document.menuform.F2.selectedIndex == 2) location =
                'Specifications_1.html';

        else if (document.menuform.F2.selectedIndex == 3) location =
                'design_question.html';

        else if (document.menuform.F2.selectedIndex == 4) location =
                'design_questions_1.html';

        else if (document.menuform.F2.selectedIndex == 5) location =
                'MNS_1.html';

}
```

```
<!------------------------------------------------------>
<!-- Name:  DEMOfrontpage.html (continued) >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard >
<!-- Purpose:  This file creates the initial user interface. >
<!--    It has four pull-down menus: REQUIREMENTS, DESIGN, >
<!--    IMPLEMENTATION, AND MAINTENANCE. >
<!--    The file was partially produced using the editor >
<!--    found in Netscape Navigator version 3.01. >
<!------------------------------------------------------>


function switch_page3() {

      if (document.menuform.F3.selectedIndex == 0) location =
            'change_request.html';

      else if (document.menuform.F3.selectedIndex == 1) location =
            'change_request.html';

      else if (document.menuform.F3.selectedIndex == 2) location =
            'Change_Requests_1.html';

      else if (document.menuform.F5.selectedIndex == 3) location =
            'bug_report.html';

      else if (document.menuform.F5.selectedIndex == 4) location =
            'Bug_Reports_1.html';

      else if (document.menuform.F5.selectedIndex == 5) location =
            'maintenance_question.html';

      else if (document.menuform.F5.selectedIndex == 6) location =
            'maintenance_questions_1.html';

      else if (document.menuform.F3.selectedIndex == 7) location =
            'MNS_1.html';

}


function switch_page4() {

      if (document.menuform.F4.selectedIndex == 0) location =
            'formpage.html';

      else if (document.menuform.F4.selectedIndex == 1) location =
            'formpage.html';

      else if (document.menuform.F4.selectedIndex == 2) location =
            'Specifications_1.html';
<!-------------------------------------------------------->
<!-- Name:  DEMOfrontpage.html (continued) >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard >
<!-- Purpose:  This file creates the initial user interface. >
<!--   It has four pull-down menus: REQUIREMENTS, DESIGN, >
<!--   IMPLEMENTATION, AND MAINTENANCE. >
<!--   The file was partially produced using the editor >
<!--   found in Netscape Navigator version 3.01. >
<!-------------------------------------------------------->


      else if (document.menuform.F4.selectedIndex == 3) location =
            'Requirements_1.html';

      else if (document.menuform.F4.selectedIndex == 4) location =
            'implementation_question.html';

      else if (document.menuform.F4.selectedIndex == 5) location =
            'implementation_questions_1.html';

      else if (document.menuform.F4.selectedIndex == 6) location =
            'MNS_1.html';
}

</SCRIPT>
</HEAD>

<! Display pull-down menus with choices (requirements, design,
implementation, and maintenance)>

<BODY LINK="#0000FF" VLINK="#800080">


<CENTER><TABLE BORDER=0 WIDTH="100%" HEIGHT="90%">
<CENTER><P><FORM WIDTH=25 NAME="menuform"></P></CENTER>
<CENTER><TABLE BORDER=1>
<TR>


<!Display "Front Loaded Accurate Requirements Engineering">
<TD><FONT COLOR="#FF0000"><FONT SIZE=+3>F</FONT></FONT><B>RONT</B>
<P><FONT COLOR="#FF0000"><FONT SIZE=+3>L</FONT></FONT><B>OADED</B></P>
<P><FONT COLOR="#FF0000"><FONT
SIZE=+3>A</FONT></FONT><B>CCURATE</B></P>
<P><FONT COLOR="#FF0000"><FONT
SIZE=+3>R</FONT></FONT><B>EQUIREMENTS</B></P>
<P><FONT COLOR="#FF0000"><FONT
SIZE=+3>E</FONT></FONT><B>NGINEERING</B><BR>
<BR>
<BR>
</P>
```

```
<!---------------------------------------------------------->
<!-- Name:  DEMOfrontpage.html (continued) >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard >
<!-- Purpose:  This file creates the initial user interface. >
<!--   It has four pull-down menus: REQUIREMENTS, DESIGN, >
<!--   IMPLEMENTATION, AND MAINTENANCE. >
<!--   The file was partially produced using the editor >
<!--   found in Netscape Navigator version 3.01. >
<!---------------------------------------------------------->



<CENTER><P><B><FONT SIZE=-2><A HREF="FLARE.html">FOR
MORE<BR>INFORMATION<BR>
CLICK HERE.</A></FONT></B></P></CENTER>
</TD>
<TD>
<CENTER><TABLE CELLSPACING=0 CELLPADDING=0 WIDTH="100%" HEIGHT="100%" >
<TR>
<TD></TD>
<TD>

<!Display pull-down menus>

<CENTER><P><SELECT NAME="F1" onChange='switch_page1();' align="left" >
<OPTION>REQUIREMENTS
<OPTION>Enter Requirement
<OPTION>View Requirements
<OPTION>Ask A Req. Question
<OPTION>View Req. Questions
<OPTION>Mission Needs Statement
</SELECT></P></CENTER>
</TD><TD></TD></TR><TR><TD>

<DIV ALIGN=right><P><IMG SRC="rightarrow.gif" HEIGHT=87
WIDTH=85></P></DIV></TD><TD></TD>

<TD><IMG SRC="leftarrow.GIF" HEIGHT=87 WIDTH=85></TD></TR>

<TR><TD>
<CENTER><P><SELECT NAME="F2" onChange='switch_page2();' align="center">

<OPTION>DESIGN
<OPTION>Enter Specification
<OPTION>View Specifications
<OPTION>Ask A Design Question
<OPTION>View Design Questions
<OPTION>Mission Needs Statement
</SELECT></P></CENTER>
</TD>
```

```
<!------------------------------------------------------------>
<!-- Name:  DEMOfrontpage.html (continued) >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard >
<!-- Purpose:  This file creates the initial user interface. >
<!--   It has four pull-down menus: REQUIREMENTS, DESIGN, >
<!--   IMPLEMENTATION, AND MAINTENANCE. >
<!--   The file was partially produced using the editor >
<!--   found in Netscape Navigator version 3.01. >
<!------------------------------------------------------------>



<TD>
<CENTER><P><A HREF="All_Reports.html">
<IMG SRC="Test.gif" ALT="U.S. ARMY" HEIGHT=121 WIDTH=121
ALIGN=ABSCENTER></P></CENTER>
</TD><TD>
<CENTER><P><SELECT NAME="F3" onChange='switch_page3();' align="center">

<OPTION>MAINTENANCE
<OPTION>Enter Change Request
<OPTION>View Change Requests
<OPTION>Enter A Bug Report
<OPTION>View Bug Reports
<OPTION>Ask Maintenance Question
<OPTION>View Maintenance Questions
<OPTION>Mission Needs Statement
</SELECT></P></CENTER>
</TD>
</TR>

<TR><TD><DIV ALIGN=right><P><IMG SRC="leftarrow.GIF" HEIGHT=87
WIDTH=85></P></DIV>
</TD>

<TD></TD>
<TD><IMG SRC="rightarrow.gif" HEIGHT=87 WIDTH=85></TD></TR>

<TR><TD><CENTER><P><FONT SIZE=-1>&nbsp; </FONT></P></CENTER>
</TD><TD></TD><TD></TD></TR></TABLE></CENTER>

<CENTER><TABLE>
<TR><TD><SELECT NAME="F4" onChange='switch_page4();' align="center">

<OPTION>IMPLEMENTATION
<OPTION>Enter Estimations
<OPTION>View Specifications
<OPTION>View Requirements
<OPTION>Ask Implementation Question
<OPTION>View Implementation Questions
<OPTION>Mission Needs Statement
</SELECT></TD>
```

```
<!------------------------------------------------------------->
<!-- Name:  DEMOfrontpage.html (continued) >
<!-- Date:  29 March, 1997 >
<!-- Author:  Anthony E. Leonard >
<!-- Purpose:  This file creates the initial user interface. >
<!--   It has four pull-down menus: REQUIREMENTS, DESIGN, >
<!--   IMPLEMENTATION, AND MAINTENANCE. >
<!--   The file was partially produced using the editor >
<!--   found in Netscape Navigator version 3.01. >
<!------------------------------------------------------------->



<TD WIDTH="20" HEIGHT="20"></TD></TR>
</TABLE></CENTER></TD></TR>
</TABLE></CENTER>



<CENTER><TABLE CELLSPACING=0 CELLPADDING=0 WIDTH="80%" HEIGHT="4%"
BGCOLOR="#FFFF80" ><TR>

<TD>&quot;<B><FONT SIZE=+1><A
HREF="http://www.sei.cmu.edu/products/publications/92.reports/92.tr.012
.html">Requirements
engineering</A></FONT></B> is the disciplined application of scientific
principles and techniques for developing, communicating, and managing
requirements&quot;<FONT SIZE=-2>[p.
68]</FONT></TD>

</TR>
</TABLE></CENTER></TR>
</TABLE></CENTER>

</BODY>
</HTML>
```

LIST OF REFERENCES

1. Luqi and Goguen, J., "Formal Methods: Promises and Problems," IEEE Software, January 1997, pp. 73-85.

2. Joint Pub 0-2, Unified Action Armed Forces (UNAAF), 24 February 1995.

3. Zave, P., "Classification of Research Efforts in Requirements Engineering," in Proceedings Second IEEE International Symposium on Requirements Engineering, IEEE Computer Soc. Press, Los Alamitos, Calif., 1995, pp. 214-216.

4. Software Test & Evaluation Panel (STEP), Requirements Definition Implementation Team. "Operational Requirements for Automated Capabilities," Draft Pamphlet (Draft PAM), April 23, 1991, available from http://140.229.1.16:9000/htdocs/teinfo/directives/pub/pub63d.html; Internet: accessed 8 March 1997. Quoted in M. Christel and K. Kang, "Issues in Requirements Elicitation," Software Engineering Institute, CMU/SEI-92-TR-012, ESC-TR-92-012, September 1992.

5. Emam, K., Quintin, S. and Madhavji, N., "User Participation in the Requirements Engineering Process: An Empirical Study," Requirements Engineering (1996), Vol 1, number 1:4-26 1996, Springer-Verlag London Limited, available from http://www.mac.co.umist.ac.uk/RE/Volume-1/Issue-1/Vol-1,Issue-1,2.html; Internet; accessed 15 February 1997.

6. Booch, B., Software Engineering with Ada, 3rd edition, Benjamin/Cummings, Redwood City, California, 1994.

7. Rumbaugh, J., et al., Object-Oriented Modeling and Design. Englewood Cliffs: Prentice Hall, 1991.

8. Booch, G., "Grady Booch: Chief Scientist," available from http://www.rational.com/ot/booch_bio.html; Internet: accessed 1 April 1997.

9. Jones, C., "Object-Oriented Analysis with CASE," in Computer-Aided Software Engineering: Issues and Trends for the 1990s and Beyond, Idea Group Publishing, Harrisburg, PA., editor T. Bergin, 1993, pp. 318-359.

10. Luqi, Ketabchi, M., "A Computer-Aided Prototyping System," IEEE Computer Technology Series, Computer-Aided Software Engineering (CASE), Editor: E. Chikofsky, 1988, pp. 89-95.

11. Jacobson, I., Home Page, available from http://www.rational.com/world/bios/jacobson_bio.html Internet: accessed 10 May 1997.

12. The Requirements Engineering Specialist Group of the British Computer Society, "Other Requirements Engineering Sites," available from http://www.cs.york.ac.uk/bcs/resg/sites.htm Internet: accessed 1 April 1997.

13. Georgia Tech, "Software Engineering Related Hotlists: Requirements Engineering," available from http://www.cc.gatech.edu/computing/SW_Eng/hotlist.html; Internet: accessed 1 April 1997.

14. Man-Tak Shing, Associate Professor, Computer Science Department, Naval Postgraduate School, Monterey, CA. Interview by author, notes. Monterey, CA, 1 April 1997.

15. Berzins, V. and Luqi, Software Engineering with Abstractions, Addison-Wesley Publishing Company, Reading, MA, 1991.

16. Luqi, "The Role of Prototyping Languages in CASE," in International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing, Vol. 1, No. 2, 1994, pp. 131-149.

17. Noel, A., "Prototyping With Data Dictionaries for Requirements Analysis," Masters Thesis, Naval Postgraduate School, Monterey, California, March 1985.

18. Awad, M., Kuusela, J. and Ziegler, J., Object-Oriented Technology for Real-Time Systems, Upper Saddle River: Prentice Hall, 1996.

19. Rational Rose Inc., Unified Modeling Language version 1.0, available from http://www.rational.com/ot/uml/1.0/index.html Internet: accessed 3 April 1997.

20. Defense Information Systems Agency (DISA) Home Page, available from http://www.disa.mil Internet: accessed 3 April 1997.

21. Parnas, D, Home Page, available from http://www.crl.mcmaster.ca/SERG/parnas.homepg Internet: accessed 21 May 1997.

22. Bharadwaj, Ramesh and Heitmeyer, Constance L., "Applying the SCR Requirements Specification Method to Practical Systems: A Case Study," Presented at the 21st Software Engineering Workshop, NASA GSFC, Greenbelt MD, USA, Dec 4-5, 1996. Available from http://www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/1996/index1996-txt.html Internet: accessed 3 April 1997.

23. DISA, "Joint Requirements Analysis and Integration", available from http://www.disa.mil/D7/missum.html Internet: accessed 3 April 1997.

24. DISA, "Defense Information Infrastructure Common Operating Environment Home Page ," available from http://spider.osfl.disa.mil/dii/ Internet: accessed 3 April 1997.

25. Faulk, S., "Requirements Engineering: A Tutorial," Naval Research Lab, NRL/MR/5546—95-7775, November 14, 1995. Available from http://www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/1995/index1995.html Internet: accessed 2 April 1997.

26. Heitmeyer, Constance L., "Requirements Specifications for Hybrid Systems," Proceedings, Hybrid Systems Workshop III, Lecture Notes in Computer Science, Springer-Verlag, edited by R. Alur, T. Henzinger, and E. Sontag, 1996. Available from http://www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/1996/index1996-txt.html Internet: accessed 3 April 1997.

27. Luqi and Shing, M., "CAPS - A Tool for Real-Time System Development and Acquisition," Quarterly Review, Office of Naval Research, Vol. XLIV, No. 1, 1992, pp. 12-16.

28. Luqi, "Real-Time Constraints in a Rapid Prototyping Language," Journal of Computer Languages, Vol. 18, No. 2, Spring 1993, pp. 77-103.

29. Joint Pub 6-0. Doctrine for Command, Control, Communications, and Computer (C4) Systems Support to Joint Operations, 30 May 1995.

30. Defense Information Systems Agency, "Requirements Assessment and Interoperability Certification of C4I and AIS Equipment and Systems," available from http://babbage6.itsi.disa.mil:80/ntb/9002/9002.pdf Internet: accessed 15 April 1997.

31. Hoenig, C., *Information Technology: Best Practices Can Improve Performance and Produce Results* (Testimony, 02/26/96, GAO/T-AIMD-96-46).

32. Standish Group Int'l, Chaos 97, tech. Report, Dennis, Massachusetts, available from http://www.standishgroup.com/chaos.html; Internet: accessed 14 February 1997.

33. Secretary of Defense Memorandum, Use of Integrated Product and Process Development and Integrated Product Teams in DoD Acquisition, May 10 1995.

34. DoD 5000.2-R, "Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information Systems, 1996.

35. DoD Directive 5000.1, "Defense Acquisition," March 15 1996.

36. Secretary of Defense Memorandum, Cost as an Independent Variable, available from http://www.acq.osd.mil/dau/arcc/accelday/caiv.pdf ; Internet: accessed 16 April 1997.

37. Valetto, G. and Kaiser, G., "Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments," in Proceedings 7th International Workshop on Computer-Aided Software Engineering, IEEE Computer soc. Press, Los Alamitos, Calif., 1995, pp. 40-48.

38. Ramesh, B., Powers, T., Stubbs, C. and Edwards, M. "Implementing Requirements traceability: A Case Study," in Proceedings Second IEEE International Symposium on Requirements Engineering, IEEE Computer soc. Press, Los Alamitos, Calif., 1995, pp. 89-95.

39. Brun-Cottan F. and Wall, P., "Using Video to Re-Present the User," Communications of the ACM, Vol. 38, No. 5, May 1995, pp. 61-71.

40. Kaiya, H., Saeki, M. and Ochimizu, K., "Design of a Hyper Media Tool to support Requirements Elicitation Meetings," in Proceedings 7th International Workshop on Computer-Aided Software Engineering, IEEE Computer soc. Press, Los Alamitos, Calif., 1995, pp. 250-259.

41. CASE tool index, Queen's University in Kingston, Ontario, available from http://www.qucis.queensu.ca/Software-Engineering/tools.html Internet; accessed 22 March 1997.

42. Luqi, Berzins, V. and Yeh, R., "A Prototyping Language for Real-Time Software," IEEE Transactions on Software Engineering, Vol. 14, No. 10, October 1988, pp 1409-1423.

43. Luqi, "Software Evolution Through Rapid Prototyping," IEEE Computer, May 1989, pp. 13-25.

44. Macedonia M. and Brutzman, D., "MBone Provides Audio and Video Across the Internet," available from: ftp://taurus.cs.nps.navy.mil/pub/mbmg/mbone.html Internet; accessed 22 March 1997.

45. Progressive Networks, "Real Audio and Video," available from: http://www.real.com/rvnba.html Internet; accessed 22 March 1997.

46. Intel, "Internet Video Phone with Proshare Technology," available from: http://connectedpc.com/iaweb/cpc/iivphone/index.htm Internet; accessed 22 March 1997.

47. O'Leary, D., "The Internet, Intranets, and the AI Renaissance," Computer, January 1997, pp. 71-78.

48. Autonomy Corporation, "Autonomy Agents," available from: http://www.agentware.com Internet: accessed 23 March 1997.

49. Poor, A., "DVD and CD-ROM: 21st Century Storage," PC Magazine Online, available from: http://www.pcmag.com/features/cdrom/_open.htm Internet: accessed 23 March 1997.

50. Toshiba Corp., "A revolution is coming and it will change everything you think about Home Entertainment," available from http://www.toshiba.com/tacp/SD/javahome.html Internet: accessed 23 March 1997.

51. Computer Mail Order, advertisement, Byte, McGraw-Hill, Peterborough, NH, January 1987.

52. Turner Hall Publishing, advertisement, Byte, McGraw-Hill, Peterborough, NH, January 1987.

53. Nevada Computer, advertisement, Byte, McGraw-Hill, Peterborough, NH, January 1992.

54. Computerlane, advertisement, Byte, McGraw-Hill, Peterborough, NH, January 1997.

55. First Source International, advertisement, Byte, McGraw-Hill, Peterborough, NH, January 1997.

56. Crothers, B., "Memory Prices Creep Back Up," CNET, Inc., available from: http://www.news.com/News/Item/0,4,8426,00.html Internet: accessed 12 May 1997, quoting Handy from Dataquest, available from: http://www.dataquest.com Internet: accessed 12 May 1997.

57. NCSA, "A Beginner's Guide to HTML," available from: http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html Internet: accessed 23 March 1997.

58. Luqi, "A Graph Model for Software Evolution," in IEEE Computer Society Press tutorial, Software Merging and Slicing, collected by V. Berzins, May 1995, pp. 202-212.

59. University of Kansas, "An Instantaneous Introduction to CGI Scripts and HTML Forms," available from: http://www.cc.ukans.edu/info/forms/forms-intro.html Internet: accessed 23 March 1997.

60. Wright, M., FormMail, available from: http://www.worldwidemart.com/scripts/formmail.shtml Internet: accessed 23 March 1997.

61. Badr, S. and Luqi, "Automation Support for Concurrent Software Engineering," Proceeding of the 6th International Conference on Software Engineering and Knowledge Engineering, Jurmala, Latvia, June 1994, pp. 46-53.

62. Microsoft Corporation, available from: http://www.microsoft.com Internet: accessed 1March 1997.

63. Leonard, A., "Flare Download Site," available from: http://www.cs.nps.navy.mil/misc/flare/Readme.html Internet: accessed 14 April 1997.

64. Netscape Corporation, "JavaScript Authoring Guide," available from: http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.html Internet: accessed 16 April 1997.

65. Web Communications, "WWW Fill-Out Forms," available form: http://www.webcom.com/~webcom/html/tutor/forms Internet: accessed 16 April 1997.

66. Netscape Corporation, available from: http://home.netscape.com Internet: accessed 16 April 1997.

67. Elmasri, R. and Navathe, S. B., Fundamentals of Database Systems, The Benjamin/Cummings Publishing Company, Redwood City, CA 1994.

68. Cormen, T. H., Leiserson, C. E. and Rivest, R. L., Introduction to Algorithms, MIT Press, Cambridge. 1990

69. Quality Systems Software "Introducing Doors 3.0," available from: http://www.qssinc.com/doors/find_out.html Internet: accessed 18 April 1997.

70. West Virginia University and NASA, "The Web-Integrated Software metrics Environment (WISE)," available from http://research.ivv.nasa.gov/projects/WISE Internet: accessed 18 April 1997.

71. Rational Software Corporation, Rational Rose, available from: http://www.rational.com Internet: accessed 18 April 1997.

72. Aonix, "OjectAda for windows," available from: http://www.aonix.com/Products/Ada/factsht.html Internet: accessed 18 April 1987.

BIBLIOGRAPHY

Chairman Joint Chiefs of Staff (CJCS) MOP 77, Requirements Generation System Policies and Procedures.

CJCS Instruction 6212.01A, Compatibility, Interoperability, and Integration of Command, Control, Communications, Computers, and Intelligence Systems, June 30, 1995.

Duvall, L., "The Information Needs of Software Managers: A problem Driven Perspective," Proc.17th international Computer Software & Applications Conference, IEEE Computer soc. Press, Los Alamitos, Calif., Nov., 1993, pp. 270-276.

Easterbrook, S. and Callahan, J., "SCR as an IV&V Tool," Available from: http://research.ivv.nasa.gov/~steve/papers/SCR96/scr.html Internet: accessed 10 May 1997.

Faulk, Stuart R., "Software Requirements: A Tutorial," NRL Memo Report 5546-95-7775, November 1995.

Heitmeyer, C., Bull, A., Gasarch, C. and Labaw, B., "SCR*: A Toolset for Specifying and Analyzing Requirements,"Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95), Gaithersburg, MD, June 25-29,1995, pp. 109-122. Available from: http://www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/1995/index1995.html Internet: accessed 18 April 1997.

ISO/IEC 12207:1995. Information technology -- Software life cycle processes.

Joint Pub 0-2, Unified Action Armed Forces (UNAAF), 24 February 1995, pg IV-5.

Luqi, "Knowledge-Based Support for Rapid Prototyping," in Source Book of Applied Artificial Intelligence, McGraw-Hill, 1992.

Luqi, "Computer-Aided Prototyping for a Command-and-Control System Using CAPS," in Software Automation, Editor: D. Cooke, World Scientific Publishing, 1994.

Luqi, Shing, M., and Berzins, V., "An Automated Prototyping Tool for System Development and Acquisition, "NetFocus, No. 211, May, 1995, pp. 2-3.

Luqi and Yeh, R., "Rapid Prototyping in Software Development," in Encyclopedia of Software Engineering, John Wiley & Sons, Inc., 1994, pp. 1-21.

Mowbray, T. J., "An Introduction to CORBA," available from: http://stscols.hill.af.mil/CrossTalk/1997/feb/corba.html Internet: accessed 18 April 1996.

Office of the Secretary of Defense Memorandum, "Use of Ada," August 26, 1994.

Office of the Secretary of Defense Memorandum, "Software Acquisition Best Practices Initiative," July 8, 1994.

Office of the Under Secretary of Defense, DoD Guide to Integrated Product and Process Development. Available from: http://www.acq.osd.mil/te/survey/table_of_contents.html Internet: accessed 18 April 1997.

Paulk, M., Curtis, B., Chrissis, M. and Weber, C., "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, February 1993.

Paulk, M., Weber, C., Garcia, S., Chrissis, M. and Bush, M., "Key Practices of the Capability Maturity Model, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-25, DTIC Number ADA263432, February 1993.

Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially, June 1994. Available from: http://www.dtic.dla.mil/c3i/dsb.html Internet: accessed 18 April 1997.

Seveney, J. and Steinberg, G., "Requirements Analysis For a Low Cost Combat Direction System," Masters Thesis, naval Postgraduate School, Monterey, California, June 1990.

Software Technology Support Center, "Guidelines for Successful Acquisition and Management of Software Intensive Systems, Version 2.0" June 1996

Software Technology Support Center, "Software Configuration Management Technology Report" September 1994. Available from: http://stscols.hill.af.mil/CM/REPORT.HTML Internet: accessed 15 February 1997.

Stevens, R., and McCaskill, G., "Methods and Tools for the Interactions Between Systems and Software," available from: http://www.qssinc.com/papers/syscase3.html Internet: accessed 18 April 1997.

Under Secretary of Defense for Acquisition and Technology Memorandum, "Policy on Cost-Performance Trade-Offs," July 19, 1995.

Unfinished Voyages, Technical.Report, Standish Group Int'l, Dennis, Massachusetts, available from http://www.standishgroup.com/voyages.html Internet; accessed 21 February 1997.

Vemulapalli, C., "A Use Case FAQ(Frequently Asked Questions)," available from: http://www.unantes.univ-nantes.fr/usecase/Contributions/chandra.html Internet: accessed 2 April 1997.

Werner, M., "Database Design From Use Cases, Roles and Actors," available from http://www.unantes.univ-nantes.fr/usecase/rootExtension.html Internet: accessed 2 April 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101

3. ECJ6-NP 1
HQUSEUCOM
Unit 30400 Box 1000
APO, AE 09128

4. CDR Michael J. Holden, USN 3
Computer Science Department, Code CS/Hm
Naval Postgraduate School
833 Dyer Road
Monterey CA 93943-5118 USA

5. Valdis Berzins 1
Computer Science Department, Code CS
Naval Postgraduate School
833 Dyer Road
Monterey CA 93943-5118 USA

6. Luqi 1
Computer Science Department, Code CS
Naval Postgraduate School
833 Dyer Road
Monterey CA 93943-5118 USA

7. Ted Lewis 1
Computer Science Department, Code CS
Naval Postgraduate School
833 Dyer Road
Monterey CA 93943-5118 USA

8. David Hislop 1
US Army Research Office
PO Box 12211
Research Triangle Park NC, 27709-2211

9. David Dampier 1
Army Research Lab
115 O'Keefe Building
Attn: AMSRL-IS-C
Atlanta GA,30332-8062

10. CPT Anthony E. Leonard 2
Department of Electrical Engineering and Computer Science
United States Military Academy
West Point, NY 10996